Support Readiness Document
Jini Technology Starter Kit 1.1

Sun Microsystems, Inc.
Market Development and Developer Relations
Support Readiness Education

Support Readiness Document
Jini Technology Starter Kit 1.1

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
U.S.A.

Version: 1.1
Release Date: January 31, 2001

© 2001 by Sun Microsystems, Inc.—Printed in USA.
901 San Antonio Road, Palo Alto, CA 94303-4900

All rights reserved. No part of this work covered by copyright may be duplicated by any means—graphic, electronic or mechanical, including photocopying, or storage in an information retrieval system—without prior written permission of the copyright owner.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 (October 1988) and FAR 52.227-19 (June 1987). The product described in this manual may be protected by one or more U.S. patents, foreign patents, and/or pending applications.

TRADEMARKS: Jini, JavaSpaces, Java, HotJava, Solaris, JDBC, and Java Compiler are trademarks of Sun Microsystems, Inc. Solaris SPARC (Platform Edition) is a trademark of Sun Microsystems, Inc. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the United States and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc. UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd.

Production Note: This book was written in FrameMaker 5.5 for Solaris and Windows by Jennifer McGinn, mentored by Marvin P. Miller

Support Readiness Education SRD Template 2.1

Table of Contents

Preface ix

1.0 Jini Technology 1
- 1.1 Jini Technology Starter Kit 1
  - 1.1.1 Jini Technology Core Platform Compatibility Kit 2
- 1.2 Features, Advantages and Benefits 3
  - 1.2.1 Features, Advantages and Benefits of JavaSpaces Technology 3
  - 1.2.2 Features, Advantages and Benefits of the Jini Technology Core Platform Compatibility Kit 3
- 1.3 Changes and New Features in Version 1.1 3
- 1.4 Features or Services Not Provided 4
  - 1.4.1 Jini Technology Security Model 4
  - 1.4.2 GNU Make 4
- 1.5 Localization and Internationalization 4
- 1.6 Other Introductory Material 4
  - 1.6.1 Java Remote Method Invocation 4
  - 1.6.2 Jini Technology Glossary 5
  - 1.6.3 Key Concepts of Jini Technology 5
  - 1.6.4 Components of the Jini architecture 6

2.0 Product Distribution 6
- 2.1 Free Distribution 6
- 2.2 Binary and Source Licensing 6
  - 2.2.1 Redistribution of Binary Product 7
- 2.3 Logo and Trademark Guidelines 7

3.0 Requirements and Dependencies 7
- 3.1 System Requirements and Dependencies 8
  - 3.1.1 Development and Execution of Jini Services and Applications 8
  - 3.1.2 Hardware Platforms Supported 8
  - 3.1.3 Disk Space Requirements for Installation 8

3.1.4 Disk Space Requirements for Log Files 8
3.2 Software Requirements and Dependencies 8
3.2.1 Required Operating System Patches 8
3.2.2 Operating Systems and Versions Supported 9
3.2.3 Java Platforms supported 9
3.2.3.1 Networking Requirements 9
3.2.3.2 Multicast 9
3.2.3.3 Multicast Routing 9
3.2.3.4 Reliable Unicast 10
4.0 Compatibility with Previous Versions of Jini Technology 10
4.1 Bugs Fixed 10
4.1.0.1 BugID# 4255360, `OutgoingMulticastAnnouncement` infinite loop fixed 10
4.1.0.2 BugID# 4245874, Extraneous imports of `java.awt.Image` removed 10
4.1.1 Bugs Fixed in `com.sun.jini.lease` 10
4.1.1.1 BugID# 4224209, `LeaseRenewalManager` didn't do any renewals when duration was `Lease.FOREVER` 10
4.1.1.2 BugID# 4250151, `LeaseRenewalManager` didn't check for null listener 10
4.1.1.3 BugID# 4262048, `LeaseRenewalEvent` constructor did not set expiration field 11
4.1.1.4 `LeaseRenewalManager` did not take batching into account when calculating renewal times 11
4.1.2 Bugs Fixed in `com.sun.jini.lookup` 11
4.1.2.1 BugID# 4193558, `JoinManager` now checks that objects are Serializable 11
4.1.2.2 BugID# 4216812, `JoinManager` no longer allows you to join all lookup services by passing null as the group's argument 11
4.1.3 Bugs Fixed in `com.sun.jini.mahalo` 11
4.1.3.1 BugID# 4198664, `RMISecurityException` is no longer caught 11
4.1.4 Bugs Fixed in `com.sun.jini.reggie` 11
4.1.4.1 BugID# 4197398, `RegistrarProxy` should check that entry classes have `no-arg` constructors 11
4.1.4.2 BugID# 4225345, `CreateLookup` should support localization 11
4.1.4.3 BugID# 4226718, `NullPointerException` in `ClassItemIter.stepClass` 11
4.1.4.4 BugID# 4233684, `Throwable` is now caught when

unmarshalling or calling a listener 12
4.1.4.5 BugID# 4242282, Reggie now calls `toStub` when service object is `Remote` 12
4.1.4.6 BugID# 4250196, Destroying Reggie could sometimes hang under Solaris™ Hotspot™ 12
4.1.4.7 BugIDs# 4279020 and 4279175, Reggie no longer allows primitive types in entries 12
4.1.5 Bugs Fixed in `com.sun.jini.tool` 12
4.1.5.1 BugID# 4193561, `ClassServer` is more clear about paths not found when operating in verbose mode 12
4.2 New Features in Version 1.1 13
4.2.1 License Terms Changed 13
4.2.2 Specifications Reorganized 13
4.2.2.1 Specification Organization in the 1.0 Releases 13
4.2.2.2 Specification Organization in the 1.1 Releases 13
4.2.3 Utility Services Added to the Jini Software Kit 14
4.2.4 Command-line Syntax and System Properties Moved 14
4.2.5 Deprecated Classes and Interfaces 15
4.2.6 Outrigger Added to JSK 15
4.2.7 Changes in `makefiles` 15
4.2.8 Security Policy File Examples 15
4.3 Backward Compatibility With Other Versions 15
4.3.1 `RMID` Changes 15
4.3.2 Deprecated Classes 16
4.3.3 Database Compatibility 16
4.3.4 Example Service Starter 16
4.3.5 RMI Registry Use Deprecated 16
4.3.6 TCK Changes 16
4.3.7 Additional Information 17
4.3.8 Forward Compatibility with Other Versions 17
5.0 Downloading the TCK and the Jini Technology Starter Kit 17
5.1 Name and Size of Download Files 17
6.0 Installing the Jini Technology Starter Kit and the TCK 17
6.1 Installing the Jini Technology Starter Kit 17
6.1.1 Extracting the Starter Kit ZIP file 18
6.1.2 Examining the Starter Kit in the `jini1_1` Directory 18
6.2 Installing the TCK 18
6.2.1 Extracting the TCK ZIP file 18
6.2.2 Examining the TCK in the `jinitck1_1A` Directory 18

7.0 Key File Descriptions 19
7.1 JAR Files 19
7.1.1 service.jar 19
7.1.2 service-dl.jar 19
7.2 Security Policy Files 20
7.3 Log Files 20
7.3.1 Log Files for Contributed Service Implementations in the Starter Kit 20
7.3.2 RMI Log Files 20
8.0 Configuration 20
8.1 Starter Kit Configuration 20
8.2 TCK Configuration 21
8.2.1 Customize the TCK Configuration File for Your Environment 21
8.2.2 Utility Properties 21
8.2.3 Discovery Properties 22
8.2.3.1 net.jini.discovery.debug 22
8.2.3.2 net.jini.discovery.timeout 22
8.2.3.3 net.jini.discovery.interface 22
8.2.3.4 net.jini.discovery.announce 22
8.2.3.5 net.jini.discovery.mtu 22
8.2.3.6 net.jini.discovery.ttl 22
8.2.4 Join Properties 22
8.2.4.1 com.sun.jini.join.debug 22
8.2.5 Service Discovery Properties 23
8.2.5.1 com.sun.jini.sdm.debug 23
8.2.5.2 com.sun.jini.sdm.discardWait 23
8.2.6 Service Properties 23
8.2.6.1 java.security.policy 23
8.2.6.2 java.rmi.server.codebase 23
8.2.6.3 com.sun.jini.join.debug 24
8.2.7 System Properties Controlling Reggie 24
8.2.7.1 com.sun.jini.reggie.proxy.debug 24
8.2.7.2 com.sun.jini.reggie.unicastTimeout 24
8.2.7.3 net.jini.discovery.interface 24
8.2.7.4 net.jini.discovery.announce 24
8.2.7.5 net.jini.discovery.mtu 24
8.2.7.6 net.jini.discovery.ttl 25
8.2.8 System Properties Controlling Mahalo 25
8.2.8.1 com.sun.jini.use.registry 25
8.2.8.2 com.sun.jini.rmiRegistryPort 25
8.2.8.3 com.sun.jini.mahalo.managerName 25

8.2.8.4 com.sun.jini.mahalo.debug 25
8.2.9 System Properties Controlling Fiddler 25
8.2.9.1 java.security.policy 25
8.2.9.2 com.sun.jini.fiddler.server.debug 26
8.2.10 System Properties Controlling Norm 26
8.2.10.1 com.sun.jini.norm.debug 26
8.2.11 System Properties Controlling Mercury 26
8.2.11.1 com.sun.jini.mercury.debug 26
8.2.11.2 com.sun.jini.mercury.streamPoolSize 26
8.2.12 System Properties Controlling TransientSpace and Server JVM 27
8.2.12.1 com.sun.jini.outrigger.basicspace.reapingInterval 27
8.2.12.2 com.sun.jini.use.registry 27
8.2.12.3 com.sun.jini.rmiRegistryPort 27
8.2.12.4 com.sun.jini.outrigger.spaceName 27
8.2.12.5 com.sun.jini.outrigger.debug 27
8.2.13 System Properties Controlling TransientSpace and Client JVM 27
8.2.13.1 com.sun.jini.outrigger.debug 28
8.2.14 System Properties Controlling FrontEndSpace and Server JVM 28
8.2.14.1 com.sun.jini.outrigger.backend.vm.options 28
8.2.14.2 com.sun.jini.outrigger.java 28
8.2.14.3 com.sun.jini.outrigger.backend.gcInterval 28
8.2.14.4 com.sun.jini.outrigger.basicspace.reapingInterval 28
8.2.14.5 com.sun.jini.outrigger.debug 28
8.2.15 System Properties Controlling FrontEndSpace and Setup JVM 28
8.2.15.1 com.sun.jini.use.registry 28
8.2.15.2 com.sun.jini.rmiRegistryPort 29
8.2.15.3 com.sun.jini.outrigger.spaceName 29
8.2.15.4 com.sun.jini.outrigger.debug 29
8.2.16 Java 2 Platform Properties 29
8.2.17 RMID Properties 29
8.2.18 RMI Properties 29
8.2.18.1 Supported java.rmi Properties 30
8.2.18.2 Unsupported sun.rmi Properties 30

9.0 Administering Jini Software 30

10.0 Using Jini Software 30
10.1 Starting the Jini Software Runtime Environment 30
10.1.1 Infrastructure Services: Lookup Service and RMI Activation Daemon 30
10.1.2 Starting the Infrastructure Services 31
10.1.2.1 Verify Your Platform and Path Environment 31
10.1.2.2 Running the RMI Activation Daemon, `rmid` 31
10.1.3 Starting the Infrastructure Services 32
10.2 Running Contributed Jini Service Implementations 32
10.3 Running an Example Service Client 33
10.3.1 Lookup Service Browser Operations 33
10.3.2 Starting the Lookup Service Browser 34
10.3.2.1 Command Line Usage 34
10.3.2.2 Examples for Starting Browsers 34
10.3.3 Example Screens of the Lookup Service Browser 35
10.3.3.1 Starting the Lookup Service Browser without the `-admin` Option 35
10.3.3.2 Starting the Lookup Service Browser with the `-admin` Option 35
10.3.3.3 Displaying the Lookup Services Discovered 35
10.3.3.4 Discovering Lookup Services 36
10.3.3.5 Displaying Services Registered with a Lookup Service 36
10.3.3.6 Service Types with Default Option 37
10.3.3.7 Selecting Different Service Types 37
10.3.3.8 Displaying Attributes Types 38
10.3.3.9 Using the ServiceItem Editor 39
10.3.3.10 Administering Services 41
10.3.4 Binary and Source Code of Lookup Service Browser 42
10.3.5 Code Examples of Jini Software Classes and Interfaces in the Lookup Service Browser 42
10.4 Implementing and Running Your Own Services 43
10.5 Running the TCK 43

11.0 Uninstalling the Jini Technology Starter Kit 43
11.1 Pre-Uninstall Considerations 43
11.1.1 Estimated Time Needed to Uninstall 44
11.2 How to Uninstall the Jini Technology Starter Kit 44

12.0 Tuning and Troubleshooting 44
13.1 Downloading or Installation Feedback 44
13.2 Common User or System Administrator Problems 44

12.2.1 Missing or Misconfigured Security Policy File 44
12.2.2 Codebase Problems 44
12.3 Common User Questions 45
12.4 Product Limitations 45
13.4.1 Security 45
13.4.2 Outrigger Runs in a Single JVM 45
12.5 Error Message Guide 46
12.6 Known Issues and Their Workarounds 46
12.7 Bug Categories in Bugtraq 46
12.8 debug Properties 47

13.0 Reference Information 47
13.1 Product Information 47
13.2 Technical Documentation 48
13.2.1 Technical Articles on Distributed Computing 48
13.2.2 Specifications for the Jini Technology Core Platform 48
13.2.3 Specifications for the Jini Extended Platform 48
13.2.4 Jini Technology Books 48
13.2.5 Jini Technology Fact Sheet 48
13.2.6 Jini Technology API Documentation 48
13.2.7 RMI Resources 48
13.3 Additional References 48
13.3.1 Official Jini Technology Web site 48
13.3.2 Sun’s Logo and Trademark Guidelines 49
13.3.3 jini-users Newsgroup Posting Archive 49
13.3.4 jini-users Mailing List Subscription 49
13.3.5 Jini Technology Programming FAQ 49
13.4 Contacts 49

Preface

This document provides Support Readiness information for the Jini Technology Starter Kit 1.1. The goal of Support Readiness Documents (SRDs) is to help support engineers prepare to support Software Products and Platforms Division products. SRDs are not designed to provide comprehensive product training (see the product documentation or Sun Education for this). Instead, they focus on issues immediately relevant to support, such as installation, configuration, and common user problems.

The *Jini Technology Starter Kit 1.1 SRD* can be viewed in PostScript or PDF format. The PDF version of the document allows navigation via a table of contents frame, and the benefit of live cross references and web links. Text that is underlined and in blue, such as the URL in this paragraph, are clickable links in the PDF version of the document. (Note: page numbers in the PDF document refer to printed pages, and will not coincide with the page numbers in the PDF reader status bar.) Although the blue color and underlining appear in the PostScript version, there are no live links when viewing that version.

Typographic Conventions

This document uses the following type conventions:

- The names of commands, files, Java™ objects, Java classes, and directories are shown in `regular monospace font`.
- Text that is a placeholder to be replaced with a real name or value appears in italic type; for example: `% unzip jsdt-1.4.zip -d` *destination directory*.
- Text that you type, when shown alongside computer output such as a command prompt, is shown in **`bold monospace font`**. The marker `prompt>`, in regular monospace font, represents the actual command prompt you would see on your screen, which may vary depending on your specific environment, shell, or platform. For example: `Solaris prompt>` **`ls -l`**.
- The names of menu items, buttons, windows, and keyboard keys appear in regular font with initial capitals, such as the Enter key.
- URLs that are clickable web links in the PDF version of the document are shown in blue, underlined monospace font, as in `http://java.sun.com`. Although the blue color and underlining appears in the PostScript version, there are no live links when viewing that version.
- URLs that are not clickable web links are shown in regular monospace font, such as `jsdt://stard:5555/socket/Session/chatSession.`

- Cross-references to other sections of the document are shown in regular font but are blue and underlined, as in, See Section 1.0, "JSDT Overview." In the PDF version of the document, these are clickable links to the indicated section. Although the blue color and underlining appears in the PostScript version, there are no live links when viewing that version.
- New terms and book titles appear in *italic* type.

Support Readiness Document
Jini Technology Starter Kit 1.1

This document provides support readiness information for the Jini™ Technology Starter Kit. This document is not designed to provide comprehensive product training. Instead, it focuses on issues immediately relevant to support, such as installation, configuration, and common user problems. For pointers to other documentation and information on product training, see Section 13.0, "Reference Information."

The information contained in this Support Readiness Document (SRD) is current at the time of printing. Since SRDs are typically prepared in advance of the General Availability/Revenue Release (GA/RR) date, formerly referred to as First Customer Ship (FCS), there may be more recent or complete information available from the resources mentioned in the SRD.

1.0 Jini Technology

Jini technology is a software initiative from Sun Microsystems™, Inc. based on the Java™ 2 platform. Jini technology enables all types of services and devices to simply connect into impromptu networks, making access to and delivery of new network services as simple as plugging in a telephone. Jini technology enables these devices and applications to work together in a community that is composed without extensive planning, installation, or human intervention.

In conjunction with this release, Sun has simplified the licensing practices for Jini network technology by moving to the latest version of the Sun Community Source License (SCSL), making the use of the Jini logo optional, and eliminating the licensing fee for the Jini Compatibility Logo.

1.1 Jini Technology Starter Kit

The Jini Technology Starter Kit contains the specifications and corresponding interfaces and classes for the Jini technology core infrastructure, additional Jini technology interfaces and utilities, and contributed service implementations from Sun Microsystems. The 1.1 version builds on previous versions of the starter kit by adding a set of programming helper utility classes and Jini deployment services, as well as

integration of the JavaSpaces™ Technology Kit, which was formerly available as a separate download.

The Jini Technology Starter Kit contains the following three components:

- Jini Technology Core Platform (JCP) – Provides the core Jini technology infrastructure software. It includes the specification and corresponding interfaces and classes for lookup, distributed events, leasing, transactions, and discovery and join.
- Jini Technology Extended Platform (JXP) – Provides extended Jini technology infrastructure service interfaces and utility classes. For example, discovery utilities and entry utilities are included here. The JXP includes the specifications and corresponding interfaces and classes for these, and also includes the *JavaSpaces Service Specification*.
- Jini Software Kit (JSK) – Provides implementations of the lookup service and transaction manager service specified in the Jini Technology Core Platform (JCP). It also provides implementations of utilities classes that are useful in writing new applications and services specified in the Jini Technology Extended Platform (JXP). The JSK provided Jini technology-enabled services, including JavaSpaces technology, are also specified in the JXP.

JavaSpaces technology provides a distributed persistence and object exchange mechanism for code written in the Java™ programming language. Objects are written in *entries* that provide typed grouping of relevant fields. Clients can perform simple operations on a JavaSpaces service to write new entries, lookup existing entries, and remove entries from a space. Using these tools, you can write systems to store state, and also write systems that use flow of objects to implement distributed algorithms and let the JavaSpaces service implement distributed persistence for you.

Sun offers two implementations of a JavaSpaces service, called Outrigger, as part of the JavaSpaces Technology Kit (JSTK) available from `http://developer.java.sun.com/developer/products/jini/product.offerings.html`.

Outrigger includes two implementations of a JavaSpaces service, `TransientSpace` and `FrontEndSpace`. The `FrontEndSpace` uses an object-oriented database package to guarantee persistence of entries. The `TransientSpace` does not guarantee persistence, and operates with a lower overhead.

1.1.1 Jini Technology Core Platform Compatibility Kit

There is also a new version of the Jini Technology Core Platform Compatibility Kit (TCK) 1.1A. The TCK is available as a separate download.

The Jini Technology Core Platform Compatibility Kit (TCK) is a bundle of both code and documents that is part of a process used to qualify hardware or software based on Jini network technology (Jini technology) internal deployment. It is also used for branding, as specified by the Sun™ Community Source License 3.0/Jini Technology Specific Attachment 1.0 (SCSL3/Jini TSA 1.0) that users agree to prior to downloading the TCK and the Jini Technology Starter Kit.

1.2 Features, Advantages and Benefits

For more information on the benefits that Jini technology brings to consumers, device manufacturers and users, see the white paper titled, "*Why Jini Technology Now*" at `http://www.sun.com/jini/whitepapers/whyjininow.html`.

1.2.1 Features, Advantages and Benefits of JavaSpaces Technology

- Communication between cooperating processes in a data-driven manner
- Use of Java programming language objects as the exchanged units of data
- Scalable design which can be as efficient as direct peer-to-peer exchange
- Persistent storage of entries
- Support for distributed transactions
- Resilience in the face of partial failure through the leasing of entries
- Notification when an entry of a desired type is written

1.2.2 Features, Advantages and Benefits of the Jini Technology Core Platform Compatibility Kit

The purpose of compatibility testing is to ensure that products issued with the Jini compatibility logo are good citizens in the environment of Jini technology-enabled services and/or devices (good lookup citizens). To be a good lookup citizen, a product must interoperate with other good lookup citizens and behave in a way that doesn't cause trouble for the rest of the networks, such as swamping a network with too much traffic. The rules of good citizenship are detailed in the Jini Technology Core Platform Specification, which the Jini Technology Core Platform Compatibility Kit (TCK) is designed to enforce.

The TCK software consists of a test framework and a set of tests to verify that a Jini technology-enabled product meets the core specifications. Note that a single TCK installation can test clients, services, and lookup services.

1.3 Changes and New Features in Version 1.1

The content of `net.jini.core` is essentially unchanged. Significant product changes are addressed in detail in Section 4.1, "Bugs Fixed" and Section 4.2, "New Features in Version 1.1."

- The license terms have changed (see Section 4.2.1, "License Terms Changed").
- The specifications have been supplemented (about 40% of the 1.1 specification is new) and reorganized (see Section 4.2.2, "Specifications Reorganized").
- New helper utilities and services are included in the starter kit (see Section 4.2.3, "Utility Services Added to the Jini Software Kit").
- Manual pages have been added (see Section 4.2.4, "Command-line Syntax and System Properties Moved").
- The contents of `net.jini` have changed slightly (see Section 4.2.5, "Deprecated Classes and Interfaces").
- JSTK is no longer separate (see Section 4.2.6, "Outrigger Added to JSK").

- The make files have been changed so that most of the starter kit can be built from scratch, except those classes that rely on PSEPro (see Section 4.2.7, "Changes in `makefiles`").
- Every service now has a security policy file that can be used and modified (see Section 4.2.8, "Security Policy File Examples").
- The TCK has been greatly enhanced and completely rewritten (see Section 4.3.6, "TCK Changes").
- The GUI tool to start services has been greatly enhanced and completely rewritten (see Section 4.3.4, "Example Service Starter").

1.4 Features or Services Not Provided

Two product limitations are addressed in Section 12.4, "Product Limitations.", relating to security and Outrigger running in a single virtual machine (VM).

1.4.1 Jini Technology Security Model

Currently, Jini technology relies on the security mechanisms of the Java 2 platform. Security managers, security policy files, and class loaders govern the activities of Jini services. A new security model for Remote Method Invocation (RMI) (see the latest draft specification at `http://java.sun.com/products/jdk/rmi/rmisec-doc/intro.html`), based on JAAS (`http://java.sun.com/products/jaas/`), will be part of the 1.4 release of the Java 2 Software Development Kit (SDK). This additional security in RMI will provide opportunities for greater security to Jini services and clients.

1.4.2 GNU Make

The make files in the Jini Technology Starter Kit 1.1 source directory are in GNU make file format. In order to build binaries and documentation from the source code without any modification to those make files, you need to use the GNU make utility, which is available from the Free Software Foundation, `http://www.fsf.org/`. The build procedure has numerous dependencies on UNIX® commands, and has only been tested in the Solaris™ Operating Environment. For detailed information on building the JAR files in the Jini Technology Starter Kit, see `jini1_1/doc/build.html`.

1.5 Localization and Internationalization

The 1.1 release of the Jini Technology Starter Kit is internationalized and the specifications have been localized to Japanese.

1.6 Other Introductory Material

There are several things you'll need to know before proceeding further, including Java RMI, Jini technology terms, and key concepts discussed in the following sections.

1.6.1 Java Remote Method Invocation

Anyone supporting Jini technology needs to have an intimate understanding of RMI. In addition to the RMI Support Readiness Training, you should take a look at the latest

documents in http://java.sun.com/j2se/1.2/docs/guide/rmi/ and http://java.sun.com/j2se/1.3/docs/guide/rmi.

1.6.2 Jini Technology Glossary

For definitions of the terms used in this document, refer to the *Jini Technology Glossary* at http://www.sun.com/jini/specs/jini1.1html/glossary-title.html. The location of the glossary in the starter kit is `jini1_1/doc/glossary/jini.glossary.pdf`.

1.6.3 Key Concepts of Jini Technology

For an overview of Jini technology concepts, see the "System Overview: Key Concepts" section in Chapter 2 of the *Jini Architecture Specification*. You can view this document at http://www.sun.com/jini/specs/jini1_1.pdf. In the starter kit, you will see this document as `jini1_1/doc/specs/jini-spec/jini-spec.pdf`.

Ensure that you understand the following terms before proceeding further:

- Jini service – An entity that can be used by a person, a program, or another service. A service may be a computation, a storage device, a communication channel to another user, a software filter, a hardware device, or another user. Two examples of services are document printing and translation from one word processor format to some other.
- Jini lookup service – The Jini lookup service provides a central registry of service items, representing services, available within the Jini system. This Jini lookup service is a primary means for programs to find services within the Jini system, and is the foundation for providing user interfaces through which users and administrators can discover and interact with services in the Jini system.
- Java Remote Method Invocation (RMI) – A part of the Java programming language that extends traditional remote procedure call mechanisms. RMI allows not only *data* to be passed from VM to VM around the network, but also *full objects*, including a reference to where the RMI runtime can download any code necessary for those objects. This ability to move code around the network in an automated fashion, just by referencing a remote object, enables much of the simplicity of the Jini architecture.
- Leasing – The process of one object in a distributed system granting the use of a resource to another object in that system for a certain period of time. The duration of the lease is negotiated by the two objects when access to the resource is first requested and given. Access to all of the services in a system of Jini technology services and devices is lease-based.
- Transactions – In general, a transaction is a tool that allows a set of operations to be grouped in such a way as to make them all appear to either succeed or fail; further, the operations in the set appear from outside the transaction to occur simultaneously. In the Jini architecture, the concrete representation of a transaction is encapsulated in an object.
- Distributed events – A remote object may allow other remote objects to register interest in certain types of events, and to receive a notification of the occurrence of such an event.

1.6.4 Components of the Jini architecture

For an overview of Jini technology concepts, see the "System Overview: Key Concepts" section in Chapter 2 of the *Jini Architecture Specification*. You can view this document online at http://www.sun.com/jini/specs/jini1_1.pdf. In the starter kit, you will see this document as `jini1_1/doc/specs/jini-spec/jini.pdf`.

Ensure that you understand the following three components of the Jini architecture before you proceed to the rest of the document:

- Infrastructure – The set of components that enables building a federated system of Jini technology-enabled services and/or devices (Jini system).
- Programming model – A set of interfaces that enables the construction of reliable services, including those that are part of the infrastructure and those that join into the federation.
- Services – The Jini technology infrastructure and programming model are built to enable services to be offered and found in the network federation. These services make use of the infrastructure to make calls to each other, to discover each other, and to announce their presence to other services and users.

2.0 Product Distribution

2.1 Free Distribution

The starter kit and the TCK are available free of charge, though users need to be registered with Java™ Developer Connection (JDC). Registration is free. The Sun Community Source License 3.0/Jini Technology Specific Attachment 1.0 (SCSL3/Jini TSA 1.0) also has to be signed online in order to download the software. You can download this from:
http://www.sun.com/software/communitysource/jini/download.html.

2.2 Binary and Source Licensing

Licensing for the Jini Technology Starter Kit and the TCK is made available through the Sun Community Source License 3.0/Jini Technology Specific Attachment 1.0. The SCSL3/Jini TSA 1.0 opens the source code for the Jini technology infrastructure to the community of Jini technology licensees, who are free to use, extend, improve, and repair Jini technology by following an open process that ensures both fairness and stable evolution of the technology. Community members can add to this common body of source code while still maintaining, if they wish, proprietary implementations. However, interfaces must be published so other community members can build their own implementations.

There are two levels of participation in the SCSL3/Jini TSA 1.0:

- Research Use - Enables licensees to use the source for any non-deployment purpose, which provides a way for organizations and individuals to examine and evaluate Jini technology. Research Use licensees –- including personal and educational users agree to minimal restrictions. These include posting error corrections, publishing

their modified source code with a special Research Use notification, and conforming to certain methods for posting modifications electronically.

- Commercial Use – Intended for commercial distribution and internal deployment. Commercial Use licensees, whether deploying an implementation internally or distributing a product externally, agree to additional responsibilities that are designed to ensure their application will be compatible with others. These include conforming to certain naming conventions within the source code, publishing specifications to any extensions they have developed, keeping the programming interfaces to their extensions open, and passing the current Jini Technology Core Platform Compatibility Kit. Commercial Use licensees who distribute commercial implementations are also responsible for distributing source code only to community licensees in good standing and upgrading their code to the current version of the technology. *Fees are no longer associated with Commercial Use.*

The Research Use and Commercial Use licenses are web-based click-throughs, so that joining the licensee community is simple. For more detailed information on SCSL3/Jini TSA 1.0, see http://www.sun.com/jini/licensing/overview.html.

Once a person has become a Jini technology licensee, they can join jini.org, an emerging, self-organized community of interest groups surrounding different sorts of services, such as printing and storage. Such interest groups define, refine, and standardize interfaces for their category of service, providing useful community source code, specifications, and verification suites for new and existing community members to use. For more information on jini.org, see http://www.jini.org/.

2.2.1 Redistribution of Binary Product

Please refer to the copy of the Sun Community Source License, Version 3.0, Jini Technology Specific Attachment 1.0 (SCSL3/Jini TSA 1.0). The complete license, referred to as “SCSL 3”, can be found at http://www.sun.com/jini/licensing/SCSL3_JiniTSA1.html.

The pertinent sections in the SCSL 3 are III.c and III.d, and in the Commercial Use Supplement General Terms (“CUSupp”) sections III.A.h and III.A.i.

2.3 Logo and Trademark Guidelines

The term Jini and the Jini compatibility logo are trademarks of Sun Microsystems, Inc. Sun has developed guidelines regarding its trademark for the term Jini and the Jini compatibility logo. For detailed information, see http://www.sun.com/policies/trademarks/.

3.0 Requirements and Dependencies

Version 1.1 of the Jini Technology Starter Kit depends on the Java™ 2 SDK, version 1.2.2 or later. Because Jini technology runs over RMI, the same set of product dependencies identified for RMI also applies to Jini technology. For RMI requirements and dependencies, see Section 3.0 of the *RMI Support Readiness Document*.

3.1 System Requirements and Dependencies

3.1.1 Development and Execution of Jini Services and Applications

Each system used for development of Jini services, clients, or applications must have enough disk space to install Java™ 2 SDK, Standard Edition, v1.2.2 or later and the Jini Technology Software Kit. For the actual file size of the Jini Technology Software Kit, see Section 5.1, "Name and Size of Download Files". Developers are not required to use any third party hardware or software tools in order to develop Jini services, clients, or applications.

Each system used for running Jini services and applications must have enough storage space for Java Runtime Environment (JRE) 1.2 and enough memory space for downloaded class files, the size of which would vary depending on the application.

3.1.2 Hardware Platforms Supported

Jini clients and services are built on and run on the Java 2 platform. Any hardware platform that runs a Java 2 SDK versions 1.2.2 or 1.3 should be able to install and run the starter kit and the TCK.

3.1.3 Disk Space Requirements for Installation

Table 1. Disk Space Required to Download and Install the Starter Kit and TCK

Platform	Download File Name	Download File Size	Disk Space for Installed Product	Total space required for install[1]
Solaris	JINI-1.1-G-CS.zip	7,806,795 bytes	23,560 KB	30.45 MB
Windows	JINI-1.1-G-CS.zip	7,806,795 bytes	23,560 KB	30.45 MB
Solaris	`JINITCK-1.1A-GCS.zip`	1,386,998 bytes	3,944 KB	5.17 MB
Windows	`JINITCK-1.1A-GCS.zip`	1,386,998bytes	3,944 KB	5.17 MB

1. This only includes space for the download files, the installed product, and any temporary space required during the installation process.

3.1.4 Disk Space Requirements for Log Files

Disk storage is also required on hosts that run the infrastructure services and Jini services that use logging: `rmid`, `FrontEndSpace`, Mahalo, Reggie, Norm, Mercury, and Fiddler. The size of these files varies with use.

3.2 Software Requirements and Dependencies

3.2.1 Required Operating System Patches

Any patches that are required are specific to running the Java 2 platform. At this time, the documentation for Solaris Operating Environment patches for the Java 2 platform is available at `http://java.sun.com/products/jdk/1.2/install-solaris-patches.html` and `http://java.sun.com/j2se/1.3/install-patches.html`.

3.2.2 Operating Systems and Versions Supported

The Jini Technology Starter Kit 1.1 has been tested by Sun on the following platforms:

Reference Implementation:

- Java 2 SDK, Standard Edition 1.2.2_006 Solaris Reference Implementation on the latest GA/RR versions of Solaris 2.6, Solaris 2.7, and Solaris 8 on Sparc™.

Production Releases:

- Java 2 SDK Standard Edition 1.2.2_05a Solaris Production Release for the latest GA/RR versions of Solaris 2.6, Solaris 2.7, and Solaris 8 on Sparc and Solaris 8 on X86.
- Java 2 SDK Standard Edition 1.2.2_006 for Windows 95/98 (2nd Edition)/2000/NT 4.0 Service Pack 6 (SP6).
- Java 2 Standard Edition (J2SE) 1.3.0 for Windows 95/98 (2nd Edition)/2000/NT 4.0 (SP6).
- Java 2 SDK Standard Edition 1.3.0 Solaris Production for the latest GA/RR version of Solaris 2.6, Solaris 7, and Solaris 8 on Sparc and Solaris 8 on X86.

3.2.3 Java Platforms supported

Jini clients and services depend on several new features of the Java 2 platform, which include RMI activation and a new security scheme. Therefore, Jini software will not work on any Java platforms before Java 2. As noted in the previous section, versions 1.2.2 and 1.3 of the Java 2 SDK, Standard Edition have been tested. While an earlier version of the Java 2 SDK may work, it is not explicitly supported.

As of this writing (November, 2000), neither PersonalJava™ nor EmbeddedJava™ platforms support Jini software.

For more information on Java 2 platform software, see `http://java.sun.com/j2se`.

3.2.3.1 Networking Requirements

A host platform must have a working Transmission Code Protocol/Internet Protocol (TCP/IP) stack, a hostname, an IP address, and access to a name service such as Network Information Service (NIS) or Domain Name System (DNS).

3.2.3.2 Multicast

The Jini discovery protocol is based on the multicast capability of the underlying network layer. For more detailed information on the multicast requirement of the Jini discovery protocol, see `jini1_1/doc/release-notes/jxp/discovery.html`.

3.2.3.3 Multicast Routing

Where the underlying transport is multicast IP, intelligent bridges and routers must be able to forward packets appropriately. This simply requires that they support one of the multicast IP routing protocols; most router vendors already do so.

If you are using a system that uses the Solaris Operating Environment as a router, ensure that it is configured to support multicast routing. For more detailed information on the multicast requirement of the discovery protocol, especially in the Solaris Operating Environment, see jini1_1/doc/release-notes/jxp/discovery.html.

3.2.3.4 Reliable Unicast

The Jini discovery protocol expects the underlying network layer to support a reliable unicast delivery. An example is RMI over TCP protocol.

4.0 Compatibility with Previous Versions of Jini Technology

The Jini Technology Starter Kit 1.1 is the third GA/RR release of Jini technology. Previous releases were 1.0 and 1.0.1. In addition, there were 1.1Alpha and 1.1Beta. releases.

4.1 Bugs Fixed

All of the bug fixes that went into the 1.0.1 version of the starter kit were also incorporated into the 1.1 version. All of the bug fixes listed were made in the change from 1.0 to 1.0.1, and they are also in all of the 1.1 releases:

4.1.0.1 BugID# 4255360, `OutgoingMulticastAnnouncement` infinite loop fixed

`OutgoingMulticastRequest` and `OutgoingMulticastAnnouncement` would allocate memory in an infinite loop if the data to be sent had exactly the maximum size (the code used to say `< maxPacketSize` instead of `<= maxPacketSize`).

4.1.0.2 BugID# 4245874, Extraneous imports of `java.awt.Image` removed

`Address` and `Status` had unnecessary imports of `java.awt.Image`.

4.1.1 Bugs Fixed in `com.sun.jini.lease`

4.1.1.1 BugID# 4224209, `LeaseRenewalManager` didn't do any renewals when duration was `Lease.FOREVER`

`LeaseRenewalManager` computed the expiration time as `duration` + `System.currentTimeMillis()` which could overflow if `duration` was large enough. The resulting negative value prevented renewals from occurring. The new code sets the expiration time to `Lease.FOREVER` in this case.

4.1.1.2 BugID# 4250151, `LeaseRenewalManager` didn't check for null listener

When sending a lease expiration notification, `LeaseRenewalManager` dereferenced the listener without first checking to see if it was `null`, and a `NullPointerException` resulted if it was `null`.

4.1.1.3 BugID# 4262048, `LeaseRenewalEvent` constructor did not set expiration field

The expiration field of `LeaseRenewalEvent` was not properly initialized by the constructor, so calls to `getExpiration`() returned an invalid value.

4.1.1.4 `LeaseRenewalManager` did not take batching into account when calculating renewal times

If you had *N* leases that could all be batched and had they all approximately the same expiration time, the `LeaseRenewalManager` (LRM) calculated that it would take `RENEW_RTT` time to renew each one, even though when it did actually renew them, it batched them. As *N* became large, the leases were renewed more and more prematurely. If *N* was large enough, `RENEW_RTT*N` exceeded the remaining duration of the lease, and the LRM started continuously renewing the leases in a tight loop.

4.1.2 Bugs Fixed in `com.sun.jini.lookup`

4.1.2.1 BugID# 4193558, `JoinManager` now checks that objects are Serializable

`JoinManager` used to take any object, but only `Serializable` objects would work; others would fail silently. Now the `JoinManager` constructor checks that its object argument is in fact `Serializable`.

4.1.2.2 BugID# 4216812, `JoinManager` no longer allows you to join all lookup services by passing null as the group's argument

According to `JoinManager`'s documentation, using `null` value for groups should be equivalent to using an empty array. However, `JoinManager` incorrectly joined all lookup services. The behavior has been corrected to match the documentation.

4.1.3 Bugs Fixed in `com.sun.jini.mahalo`

4.1.3.1 BugID# 4198664, `RMISecurityException` is no longer caught

`RMISecurityException` has been deprecated, so it is now ignored (where previously it was caught).

4.1.4 Bugs Fixed in `com.sun.jini.reggie`

4.1.4.1 BugID# 4197398, `RegistrarProxy` should check that entry classes have `no-arg` constructors

`ClassMapper` now checks that entry classes have `no-arg` constructors, which was a difficult-to-diagnose error otherwise.

4.1.4.2 BugID# 4225345, `CreateLookup` should support localization

`CreateLookup's` messages have been internationalized by moving them into a resource file (`reggie.properties`).

4.1.4.3 BugID# 4226718, `NullPointerException` in `ClassItemIter.stepClass`

If you called lookup with a template of type *`EntryB`*, which has no fields but extends a class *`EntryA`* which has some fields, the lookup failed with a

NullPointerException. The new code uses getDefiningClass to avoid this error.

4.1.4.4 BugID# 4233684, Throwable is now caught when unmarshalling or calling a listener

Error instances that occurred in user code supplied to Reggie were not properly caught; now they are.

4.1.4.5 BugID# 4242282, Reggie now calls toStub when service object is Remote

Reggie used to extract the service type descriptor for a Remote object that was being registered, when it should have been using the descriptor of the Remote object's stub.

4.1.4.6 BugID# 4250196, Destroying Reggie could sometimes hang under Solaris™ Hotspot™

The AnnounceThread can sometimes receive an InterruptedIOException while sending a datagram, but the exception was treated only as an IOException and discarded. As a result, the parent DestroyThread never resumed.

4.1.4.7 BugIDs# 4279020 and 4279175, Reggie no longer allows primitive types in entries

ClassMapper and LookupAttributes used to allow primitive fields in the entry types, contrary to the *Jini Entry Specification.*

4.1.5 Bugs Fixed in com.sun.jini.tool

4.1.5.1 BugID# 4193561, ClassServer is more clear about paths not found when operating in verbose mode

ClassServer, when started with the -verbose flag, would indicate failed requests as follows:

```
reggie-dl.jar from replica:38146
(not found)
```

This format was internally consistent (the first line records the access request, and the second records that it failed), but was very confusing. The new format makes the meaning of each line more clear:

```
reggie-dl.jar requested from replica:38146
reggie-dl.jar not found
```

ClassServer has been internationalized

The ClassServer now obtains all of its error messages and verbose messages from a resource bundle (com.sun.jini.tool.resources.classserver.properties).

ClassServer now has a fileDownloaded method

In order to facilitate subclasses of com.sun.jini.tool.ClassServer, a protected method has been added which is called whenever a file is successfully downloaded by a client.

`ClassServer` now writes to `System.err`
Error messages used to be written to `System.out`.

4.2 New Features in Version 1.1

4.2.1 License Terms Changed
Fees are no longer associated with Commercial Use. See Section 2.0, "Product Distribution".

4.2.2 Specifications Reorganized

4.2.2.1 Specification Organization in the 1.0 Releases
In the 1.0 release of the starter kit, there were twelve separate specifications. The eight specifications that described the core platform (the JCP) were:

- *Jini™ Architecture Specification*
- *Jini™ Device Architecture Specification*
- *Jini™ Discovery and Join Specification*
- *Jini™ Distributed Event Specification*
- *Jini™ Distributed Leasing Specification*
- *Jini™ Entry Specification*
- *Jini™ Lookup Service Specification*
- *Jini™ Transaction Specification*

The 1.0 release also had four specifications that described the extended platform, the JXP:

- *Jini™ Discovery Utilities Specification*
- *Jini™ Entry Utilities Specification*
- *Jini™ Lookup Attribute Schema Specification*
- *JavaSpaces™ Specification*

4.2.2.2 Specification Organization in the 1.1 Releases
The Jini™ Technology Core Platform Specification, version 1.1 has combined the following (formerly separate) Jini specifications into a single physical and logical unit:

- *Discovery and Join*
- *Entry*
- *Distributed Leasing*
- *Distributed Events*
- *Transaction*
- *Lookup Service*

Because the architecture specification and the device architecture specification were supplemental to the core specification, they were left as separate entities:

- *Jini™ Architecture Specification*
- *Jini™ Device Architecture Specification*

There were several new specifications that described both additions to the extended platform and some utilities that were in the 1.0 release that didn't have formal specifications. These new specs were organized together with the existing specs into *A Collection of Jini™ Technology Helper Utilities and Services Specifications*, v1.1, which contains:

- *Introduction to Helper Utilities and Services*
- *Jini™ Discovery Utilities Specification*
- *Jini™ Entry Utilities Specification*
- *Jini™ Lease Utilities Specification*
- *Jini™ Join Utilities Specification*
- *Jini™ Service Discovery Utilities Specification*
- *Jini™ Lookup Attribute Schema Specification*
- *Jini™ Lookup Discovery Service Specification*
- *Jini™ Lease Renewal Service Specification*
- *Jini™ Event Mailbox Service Specification*

Because the *JavaSpaces™ Service Specification* doesn't describe a "helper" service, but rather a service that is built using the Jini architecture, it was kept separate from the collection.

4.2.3 Utility Services Added to the Jini Software Kit

Three new utility services are specified in the Jini Technology Extended Platform (JXP): a lease renewal service, an event mailbox service, and a lookup discovery service. Each of these services is implemented in this release of the starter kit, under the names Norm, Mercury, and Fiddler, respectively. Release notes for these services can be found in `jini1_1/doc/release-notes/jsk/`.

4.2.4 Command-line Syntax and System Properties Moved

The 1.1 release of the starter kit contains manual pages for each of the service implementations: Reggie, Mahalo, Norm, Outrigger, Mercury, and Fiddler in `jini1_1/doc/manpages/`:

- Reggie – `jini1_1/doc/manpages/reggie.html`
- Mahalo – `jini1_1/doc/manpages/mahalo.html`
- Norm – `jini1_1/doc/manpages/norm.html`
- Fiddler – `jini1_1/doc/manpages/fiddler.html`
- Mercury – `jini1_1/doc/manpages/mercury.html`
- Outrigger, transient -`jini1_1/doc/manpages/transientspace.html`
- Outrigger, persistent -`jini1_1/doc/manpages/frontendspace.html`

In addition, there is now a single service properties page that lists all of the properties controlling the services in the JSK at jini1_1/doc/release-notes/jsk/serviceproperties.html.

4.2.5 Deprecated Classes and Interfaces

The following classes and interfaces were experimental in the 1.0 release and have been replaced with corresponding members in the net.jini packages. These classes and interfaces in com.sun.jini are being marked deprecated in this release and will be removed in a future release of the starter kit. Update your code to use the new versions.

- com.sun.jini.discovery.LookupLocatorDiscovery has been deprecated and replaced by net.jini.discovery.LookupLocatorDiscovery.
- com.sun.jini.lease.LeaseRenewalManager has been deprecated and replaced by net.jini.lease.LeaseRenewalManager.
- com.sun.jini.lease.LeaseListener has been deprecated and replaced by net.jini.lease.LeaseListener.
- com.sun.jini.lease.LeaseRenewalEvent has been deprecated and replaced by net.jini.lease.LeaseRenewalEvent.
- com.sun.jini.lookup.JoinManager has been deprecated and replaced by net.jini.lookup.JoinManager.
- com.sun.jini.lookup.ServiceIDListener has been deprecated and replaced by net.jini.lookup.ServiceIDListener.

4.2.6 Outrigger Added to JSK

Outrigger is now being distributed as part of the starter kit, instead of separately in the JavaSpaces Technology Kit. The intent of this change was to make JavaSpaces technology more accessible, and to enhance the value of the starter kit.

4.2.7 Changes in makefiles

We have changed the makefiles distributed with the starter kit so that most of Outrigger can be built from source code, even without eXcelon Corporation's ObjectStore PSEPro for Java development tools. See the page *Building the Jini Technology Packages from Source Code* at jini1_1/doc/build.html for details.

4.2.8 Security Policy File Examples

Because the configuration of security policy files can be complex, there is a new directory that contains policy files for each service in jini1_1/policy/. These files may be used unmodified in some cases, but generally will have to be edited to correctly specify the directory on your system where the starter kit is installed.

4.3 Backward Compatibility With Other Versions

4.3.1 RMID Changes

By default, rmid now requires a security policy file. While this change is not specific to the TCK or the starter kit, it affects the use of both the TCK and the starter kit. For a complete discussion on this change, refer to jini1_1/doc/release-notes/jsk/execpolicy.html for details.

You can also refer to the Solaris Operating Environment manual pages for `rmid` at http://java.sun.com/products/jdk/1.2/docs/tooldocs/solaris/rmid.html and http://java.sun.com/j2se/1.3/docs/tooldocs/solaris/rmid.html

and the Microsoft Windows platform manual pages for `rmid` at http://java.sun.com/products/jdk/1.2/docs/tooldocs/win32/rmid.html and http://java.sun.com/j2se/1.3/docs/tooldocs/win32/rmid.html.

4.3.2 Deprecated Classes

The Jini Technology Starter Kit 1.1 contains some Java packaging changes from the 1.0 and 1.01 releases. These changes were made to promote some classes and interfaces from `com.sun.jini` packages to `net.jini` packages, thereby moving them from the JSK into the JXP. As a result, there have been some changes in the organization of the JAR files. These new versions of the older utilities have new API's and added functionality. In order to maintain compatibility with previous versions of the starter kit, the older versions of these utilities still exist but are deprecated, so you'll see deprecation warnings when compiling with the older versions.

4.3.3 Database Compatibility

The JavaSpaces Technology Kit was incorporated into the 1.1 version of the starter kit. In order to fix certain bugs the format of the persistent store used by the 1.1 GA/RR release of `FrontEndSpace` was changed in a way that is incompatible with the database formats used in previous releases (1.0, 1.0.1, 1.1 Beta, 1.1 Beta2). Therefore, it will not be possible to upgrade a running `FrontEndSpace` service simply by killing `rmid`, replacing Outrigger's JAR files, and restarting `rmid`. Old data must be transferred to the new space if it is to be retained.

4.3.4 Example Service Starter

The new application, `com.sun.jini.example.launcher.StartService` replaces `com.sun.jini.example.service.StartService`. The older version did not have any way to save its state and could only be extended by adding more code. This new version tries to make easier the execution of Jini services and the infrastructure services they require.

4.3.5 RMI Registry Use Deprecated

Using the `rmiregistry` to find Outrigger and Mahalo services has been deprecated. The use of the `rmiregistry` was a temporary measure that predates the public release of the Jini lookup service (a Beta version of JavaSpaces was released for the JavaOne conference in 1998). New services that have been developed for the starter kit since the first public release of the Jini lookup service in January of 1999 use the Jini lookup service exclusively.

4.3.6 TCK Changes

The TCK was completely rewritten between the 1.0 and 1.0B releases. All of the changes from 1.0 to 1.0B as well as the changes from 1.0B to 1.1A are documented in the file `jinitck1_1A/doc/relnotes.html`. While the only change from 1.0B to 1.1A was that Admins were provided for the three new services (Norm, Mercury, and

Fiddler), there is no compatibility between the original 1.0 release and the two later releases (1.0B and 1.1A).

4.3.7 Additional Information

For more information on changes from the Beta releases, see the following release notes:

- `jini1_1/doc/release-notes/jcp.html`
- `jini1_1/doc/release-notes/jxp.html`
- `jini1_1/doc/release-notes/jsk.html`

4.3.8 Forward Compatibility with Other Versions

The `com.sun.jini` package and its subpackages contain the contributed implementations of Jini services from Sun Microsystems. At present, all classes and interfaces in these packages are subject to change or removal in a future release, in ways that may not be backward compatible with this release.

5.0 Downloading the TCK and the Jini Technology Starter Kit

You can download the Jini Technology Starter Kit and the TCK from `http://www.sun.com/software/communitysource/jini/download.html`. You'll need to be registered with Java™ Developer Connection (JDC); registration is free. The Sun Community Source License 3.0/Jini Technology Specific Attachment 1.0 (SCSL3/Jini TSA 1.0) also has to be signed online in order to download the software.

5.1 Name and Size of Download Files

- Jini Technology Starter Kit (`JINI-1.1-G-CS.zip`) – Includes the binaries, source code, specifications, and support documentation (7,806,795 bytes)
- Jini Technology Core Platform Compatibility Kit (`JINITCK-1.1A-G-CS.zip`) – Includes the TCK binaries, harness and test specifications, and support documentation (1,386,998 bytes)

6.0 Installing the Jini Technology Starter Kit and the TCK

There are currently two Jini technology distribution bundles:

- `JINI-1.1-G-CS.zip` – The starter kit binaries, source code, and docs
- `JINITCK-1.1A-G-CS.zip` – The TCK binaries, source code, and docs

6.1 Installing the Jini Technology Starter Kit

Once the starter kit has been extracted, you can access the information in this section from `jini1_1/doc/installation.html`. In addition, this information is provided

at http://developer.java.sun.com/developer/products/jini/installation.index.html

6.1.1 Extracting the Starter Kit ZIP file

The starter kit extracts into the directory, `jini1_1`. To extract the downloaded ZIP file:

- If you have the Java 2 SDK installed, you can use the `jar` command:

```
prompt> jar xvf JINI-1.1-G-CS.zip
```

- On a UNIX system, you can use the UNIX version of unzip:

```
prompt> unzip JINI-1.1-G-CS.zip
```

- On Microsoft Windows systems, you can use the unzipping tool of your choice.

6.1.2 Examining the Starter Kit in the `jini1_1` Directory

- `index.html` – The starting page for all documentation
- `doc` directory – Contains API documentation, supporting documentation for the examples, installation instructions, license file(s), glossary and release notes
- `example` directory – Contains supporting files for running the example code
- `lib` directory – Contains the JAR files
- `source` directory – Contains the source code for the binary classes
- `policy` directory - Contains example security policy files

6.2 Installing the TCK

Once the starter kit has been extracted, you can access the information in this section from `jinitck1_1A/doc/installation.html`. In addition, this information is provided at http://developer.java.sun.com/developer/products/jini/installation_tck1_1A.html.

6.2.1 Extracting the TCK ZIP file

After you download the TCK 1.01A you need to extract it. For example,

The TCK extracts into the `jinitck1_1A` directory. To extract the downloaded ZIP file:

- If you have the Java 2 SDK installed, you can use the `jar` command:

```
prompt> jar xvf JINITCK-1.1A-G-CS.zip
```

- On a UNIX system, you can use the UNIX version of unzip:

```
prompt> unzip JINITCK-1.1A-G-CS.zip
```

- On Microsoft Windows systems, you can use the unzipping tool of your choice.

6.2.2 Examining the TCK in the `jinitck1_1A` Directory

- `index.html` – The starting page for all documentation
- `config` directory - Contains security policy and properties files

- `doc` directory – Contains API documentation, directions for running the TCK, installation instructions, test and harness specifications, license file(s), a document on passing the TCK, and release notes
- `lib` directory – Contains the JAR files
- `source` directory – Contains the source code for the binary classes

7.0 Key File Descriptions

7.1 JAR Files

In the `lib` directory of the starter kit and the TCK are "binary" or "executable" files, distributed as JAR files, that contain compiled Java programming language (`.class`) files.

Note: The rest of the information in this section can be found at `jini1_1/doc/release-notes/jsk.html#jars`.

To facilitate code downloading to clients, the implementation of every service implementation in the starter kit is separated into two (and in the case of Fiddler, three) JAR files. For this discussion, service can refer to any of the services in the starter kit: Reggie, Mahalo, Norm, Mercury, Outrigger, or Fiddler.

7.1.1 `service.jar`

This file contains all of the class files needed by the service in order to run. This file has been configured as an executable JAR. When used with the `-jar` command-line option in the Java Virtual Machine (JVM), it will start up the service. This file must be in the CLASSPATH of the service. At times, you may see CLASSPATH referred to as the service's import codebase.

7.1.2 `service-dl.jar`

This file contains all of the class files that clients need to download (dl stands for download) from the service's codebase in order to interact with the service. The `java.rmi.server.codebase` property of the service must be a URL that points to the `service-dl.jar` file. If the codebase property has been properly set, the `service-dl.jar` file does not have to be a part of the client's CLASSPATH because the code will be downloaded from the URL specified in the codebase. This allows client programs to be deployed independently from any particular service implementation. Whereas the service's CLASSPATH is sometimes referred to as the service's import codebase, the value of the service's `java.rmi.server.codebase` property can be referred to as the service's export codebase. For more information on the `java.rmi.server.codebase` property, you can refer to the tutorial, *Dynamic Code Downloading Using the java.rmi.server.codebase Property* at `http://java.sun.com/j2se/1.3/docs/guide/rmi/codebase.html`.

Note: A URL may refer to a remote machine, local machine, or local file system.

7.2 Security Policy Files

A security policy file must be constructed and maintained for each service in the starter kit. Because this process can be confusing and difficult, the 1.1 version of the starter kit contains sample policy files that can be used as-is or with just slight modification. These policy files can be found in the `jini1_1/policy/` directory.

The Java 2 SDK documentation describes the structure and use of security policy files:

http://java.sun.com/products/jdk/1.2/docs/guide/security/PolicyFiles.html
http://java.sun.com/products/jdk/1.2/docs/guide/security/permissions.html

7.3 Log Files

7.3.1 Log Files for Contributed Service Implementations in the Starter Kit

Each persistent service in the starter kit creates a log file that maintains the state of that service so that it can be restored after system crash or reboot. The path of the log file is specified as an option when the service gets started. The file is in binary file form and the meta data describing the structure of its contents are not in public domain.

7.3.2 RMI Log Files

The `rmid` command uses a `log` directory to store information about registered objects across reboots and other events. This log is read during `rmid` startup and frequently updated in a stable manner. The file is in binary file form and the meta data describing the structure of its contents are not in the public domain. On UNIX, you can use the `strings` command to determine how services have been registered with the `rmid` command tool.

8.0 Configuration

8.1 Starter Kit Configuration

As discussed earlier, the contributed lookup service implementation, Reggie, requires that `rmid` and an HTTP server be running. Each of the contributed service implementations requires that a Jini lookup service be running. The normal case is that a lookup service is available on the network, and it will be found using the Jini discovery protocol.

There are many properties that control the behavior of Jini services and clients. These include utility and service properties that are part of the code in the starter kit, properties that control the JVM in which the service or client runs, properties that control how `rmid` behaves, and properties that affect the underlying RMI runtime system.

8.2 TCK Configuration

While complete instructions for running and configuring the TCK are included in the file, jinitck1_1A/doc/running.html, the steps to modify the property file for the test runs of the TCK against Mahalo are provided here for your convenience.

8.2.1 Customize the TCK Configuration File for Your Environment

1. Change directory to the config subdirectory of the TCK installation directory and copy the `tck.prop` file to a new file name, for modification. This file name will be passed to the TCK command line later. None of the files provided with the TCK distribution should be modified directly. There are at least two property values, specifying directory locations, that you will need to set properly in order for the TCK tests to run as they should.

   For example, on a UNIX system where you have the TCK installed in `/test/jinitck1_1A/`

   ```
   % cd /test/jinitck1_1A/config
   % cp tck.prop my.prop
   ```

   Or on the Microsoft Windows NT platform, where you have the TCK software installed in `C:est\jinitck1_1`:

   ```
   % cd C:\test\jinitck1_1A\config
   % copy tck.prop my.prop
   ```

2. Edit the configuration file

   For our first example run of the TCK, we'll run the Mahalo service without using the provided admin interface, so we'll need to change three properties in the config file: `com.sun.jini.tck.scratchDir`, `com.sun.jini.tck.installDir`, and `com.sun.jini.tck.unofficialQuietTime`. The impacts of changing `com.sun.jini.tck.unofficialQuietTime` to override its default value of 30 minutes are that we'll speed up the TCK execution time, and it will be considered an "unofficial" run of the TCK. Unofficial runs of the TCK are discussed in the *Jini™ Technology Core Platform Compatibility Kit Harness Specification*.

   For the second example run of the TCK, we'll use the admin for Mahalo that is provided with the TCK, so we'll need to uncomment and change an additional four properties.

   For now though, you'll need to edit your copy of the config file, `my.prop`, so that the `com.sun.jini.tck.installDir` accurately reflects the installation directory of the TCK on your system. Next, you'll need to change the `com.sun.jini.tck.scratchDir` to a location to which the TCK may write temporary files and log files. The directory that you specify will be created for you as part of the TCK run. Lastly, you'll want to change the value of `com.sun.jini.tck.unofficialQuietTime` from 1800000 (milliseconds) to 60000.

8.2.2 Utility Properties

This information is excerpted from jini1_1/doc/release-notes/jxp/utilityproperties.html.

Note: Unless otherwise stated, any output from these properties is sent to `System.out`.

8.2.3 Discovery Properties

8.2.3.1 `net.jini.discovery.debug`

If set to any value, this property indicates that debugging information related to the execution of the Jini discovery protocols should be printed to `System.err`.

8.2.3.2 `net.jini.discovery.timeout`

When a client or service uses the unicast request protocol to communicate with a particular lookup service, the socket connection used by the protocol will time out after *N* milliseconds, where *N* is defined by this property. The default value is 60,000 milliseconds (1 minute).

8.2.3.3 `net.jini.discovery.interface`

When this property is set to either a host name or IP address, the network interface corresponding to the given name or address will be used in the multicast discovery process. When not set, the network interface used is typically the default interface as determined by the underlying operating system.

Note: The `net.jini.discovery.interface` property can also be set on the JVM in which the Reggie implementation of the lookup service runs. The value of this property controls the network interface the Reggie implementation uses when participating in the multicast discovery protocol as both an entity that is discovered by other entities, a lookup service, and as an entity that discovers other lookup services.

8.2.3.4 `net.jini.discovery.announce`

A Jini lookup service will send out multicast packets announcing its existence every *N* milliseconds. Currently, the default value of this property is 120,000 milliseconds (two minutes). The value of this property can be used to control how often the Reggie implementation of the lookup service sends out multicast announcements and, for entities that discover lookup services, it can be used to control how often multicast announcements from previously discovered lookup services are examined for liveness.

8.2.3.5 `net.jini.discovery.mtu`

If set, this property must be an integer between 512 and the Maximum Transmission Unit (MTU) allowed by your system. The value of this property specifies the maximum permissible size of multicast request and announcement packets. The default value is 512.

8.2.3.6 `net.jini.discovery.ttl`

If set, this property must be an integer between 1 and 255. This property specifies the time-to-live parameter used by both the multicast announcement and multicast request protocol implementations. The default value is 15.

8.2.4 Join Properties

8.2.4.1 `com.sun.jini.join.debug`

If set to any value, this property indicates that debugging information related to the execution of the Jini join protocols should be printed to `System.err`. This property can

be set on the JVM of any entity that employs the join utility, `net.jini.lookup.JoinManager`.

8.2.5 Service Discovery Properties

8.2.5.1 `com.sun.jini.sdm.debug`

If set to any value, this property indicates that debugging information related to exceptional conditions that occur during service discovery processing will be printed to `System.err`. This property should be useful in diagnosing common service discovery problems related to a bad listener codebase and/or a non-existent or poorly defined proxy equals method, as well as other problems where exceptions are swallowed.

8.2.5.2 `com.sun.jini.sdm.discardWait`

This property can be used to effect the behavior of the mechanism that handles the service discard problem described in the *Jini Service Discovery Utilities Specification*. This property allows each entity that uses the service discovery manager to define how long (in milliseconds) to wait for verification from the lookup service(s) that a discarded service is actually down before committing or uncommitting a requested service discard. The current implementation of the service discovery manager defaults to waiting 10 minutes (twice the maximum lease duration granted by the Reggie implementation of the lookup service).

8.2.6 Service Properties

There are a few properties that apply to each of the services in the starter kit.

8.2.6.1 `java.security.policy`

The security policy file that is required for each contributed service implementation in the starter kit will be different. Because using a policy file that grants all permissions is not recommended and because writing a security policy file that fits all the needs of a service can be difficult, sample security policy files are supplied with the 1.1 version of the starter kit in the `jini1_1/policy/` directory. These policy files can be used as-is or with little customization.

8.2.6.2 `java.rmi.server.codebase`

The codebase property may well be the most misunderstood property in the history of the Java programming language. While this is no fault of the RMI developers, the misunderstanding derives from the fact that the codebase property not only affects the VM on which it is set, but it also affects the behavior of the remote VM that uses it. In other words, it not only affects the remote object's (read that as service's) VM, but the VM of every client that uses that remote object. For a tutorial on the codebase property, see `http://java.sun.com/j2se/1.3/docs/guide/rmi/codebase.html`.

For the purpose of this document, the codebase property should be set to the URL location of the downloadable (`-dl.jar`) client files. The HTTP server that you start is used to serve up these JAR files. Make sure that the web root directory of the HTTP server contains all of the `-dl.jar` files that are in the `jini1_1/lib/` directory. These files represent the code that will download to clients of the services in the starter kit.

8.2.6.3 com.sun.jini.join.debug

The current implementations of each of the services in the starter kit participates in the Jini discovery and join protocols using the `LookupDiscovery` class. The operation of this class can be controlled by the system properties described in Section 8.2.2, "Utility Properties." In addition, each of these services may use the `com.sun.jini.join.debug` property. If these properties need to be used, they can be set using `-D`, at the end of the command line.

Note: The following information is excerpted from `jini1_1/doc/release-notes/jsk/serviceproperties.html`. For complete descriptions of these properties, refer to the service properties document, or to that services manual page under `jini1_1/doc/manpages/`. Unless otherwise stated, any output from these properties is sent to `System.out`.

8.2.7 System Properties Controlling Reggie

8.2.7.1 com.sun.jini.reggie.proxy.debug

If set to any value, this property indicates that debugging information related to the execution of Reggie's proxy should be printed to `System.err`. Set this property on clients, if calls to the lookup service are unexpectedly returning null values for service objects, service types, attribute set classes, attribute sets, or attribute values.

8.2.7.2 com.sun.jini.reggie.unicastTimeout

When an instance of Reggie uses the unicast request protocol to communicate with a client or service attempting to discover that lookup service, the socket connection used by the protocol will time out after *N* milliseconds, where *N* is defined by this property. Currently, the default value of this property is 60,000 milliseconds (one minute).

8.2.7.3 net.jini.discovery.interface

When this property is set to either a host name or IP address, the network interface corresponding to the given name or address will be used by Reggie when participating in the multicast (group) discovery protocol as both an entity that is discovered by other entities (a lookup service), and as an entity that discovers other lookup services.

8.2.7.4 net.jini.discovery.announce

A Jini lookup service will send out multicast packets announcing its existence every *N* milliseconds. The value of this property can be used to control how often the Reggie implementation of the lookup service sends out multicast announcements. For entities that discover lookup services, it can be used to control how often multicast announcements from previously discovered lookup services are examined for liveness.

8.2.7.5 net.jini.discovery.mtu

If set, this property must be an integer between 512 and the Maximum Transmission Unit (MTU) allowed by your system. The value of this property specifies the maximum permissible size of multicast request and announcement packets. The default value is 512.

8.2.7.6 net.jini.discovery.ttl

If set, this property must be an integer between 1 and 255. This property specifies the time-to-live parameter used by both the multicast announcement and multicast request protocol implementations. The default value is 15.

8.2.8 System Properties Controlling Mahalo

8.2.8.1 com.sun.jini.use.registry

If the `com.sun.jini.use.registry` property is set to a non-null value, the service will bind itself to an RMI registry. This property must be set on the setup JVM. It is only used when creating a Mahalo service from the command line. This property has no effect on how Mahalo binds itself to Jini lookup services.

Note: Mahalo's support for binding to an RMI registry has been deprecated and will be removed in future versions of Mahalo. The Jini lookup service is the preferred method for advertising a service.

8.2.8.2 com.sun.jini.rmiRegistryPort

If the RMI registry is being used, this property can be used to set the port on which the service will look for the registry. This property must be set on the setup JVM. It is only used when creating a Mahalo service from the command line.

8.2.8.3 com.sun.jini.mahalo.managerName

This property defines the name associated with the transaction manager. If the transaction manager is bound to an RMI registry, then the value of the `managerName` property is the name to which it is bound. If the transaction manager is bound to a Jini lookup service, this name is used to create an attribute of type `net.jini.lookup.entry.Name` that is associated with the transaction manager's `ServiceItem`. This property must be set on the setup JVM. It is only used when creating a Mahalo service from the command line.

8.2.8.4 com.sun.jini.mahalo.debug

If set, this property enables the output of debugging information. This property can be set on both the setup JVM and the server JVM. The comma-separated list in the value of the property specifies which subsystems should generate debugging information, and to which output stream that information should be sent. By default, messages will go to `System.out` unless an optional log file is provided on a per-subsystem basis. See the manual page at `jini1_1/doc/manpages/mahalo.html` or `jini1_1/doc/release-notes/jsk/serviceproperties.html` for a complete description of how to use this property.

8.2.9 System Properties Controlling Fiddler

8.2.9.1 java.security.policy

When starting Fiddler from the command line, the `java.security.policy` can be set as part of the `setup_jvm_options` to define which security policy file should be used for the setup JVM, but under normal circumstances, this setting will not be necessary for the setup JVM. See the manual page at `jini1_1/doc/manpages/fiddler.html` or

jini1_1/doc/release-notes/jsk/serviceproperties.html for a complete description of this property.

8.2.9.2 com.sun.jini.fiddler.server.debug

If set, this property enables the output of debugging information. The comma-separated list in the value of the property specifies which subsystems should generate debugging information, and to which output stream that information should be sent. Messages will go to the default output stream unless an optional log file is provided on a per-subsystem basis. See the manual page at jini1_1/doc/manpages/fiddler.html or jini1_1/doc/release-notes/jsk/serviceproperties.html for a complete description of this property.

8.2.10 System Properties Controlling Norm

8.2.10.1 com.sun.jini.norm.debug

If set, this property enables the output of debugging information. The comma-separated list in the value of the property specifies which subsystems should generate debugging information, and to which output stream that information should be sent. Messages will go to System.out by default, unless an optional log file is provided on a per-subsystem basis. See the manual page at jini1_1/doc/manpages/norm.html or jini1_1/doc/release-notes/jsk/serviceproperties.html for a complete description of this property.

8.2.11 System Properties Controlling Mercury

8.2.11.1 com.sun.jini.mercury.debug

If set, this property enables the output of debugging information. The comma-separated list in the value of the property specifies which subsystems should generate debugging information, and to which output stream that information should be sent. Messages will go to the default output stream unless an optional log file is provided (on a per-subsystem basis). See the manual page at jini1_1/doc/manpages/mercury.html or jini1_1/doc/release-notes/jsk/serviceproperties.html for a complete description of this property.

8.2.11.2 com.sun.jini.mercury.streamPoolSize

This property determines the maximum number of concurrent event log input/output streams. An existing stream is closed before opening another stream, once this number has been reached. The default value is 10.

Note: Each file-based stream will use a file descriptor. The maximum number of open file descriptors is a limiting factor on most operating systems. You should tune this parameter to suit your operating environment.

8.2.12 System Properties Controlling TransientSpace and Server JVM

8.2.12.1 com.sun.jini.outrigger.basicspace.reapingInterval

This property controls how often the active reaping of unused entry/template objects takes place in the TransientSpace. This property is set to a long that represents the time in milliseconds between reaping attempts. If not specified, a default value of five minutes is used. Reducing this value helps reduce the memory footprint of a heavily loaded space.

8.2.12.2 com.sun.jini.use.registry

If the com.sun.jini.use.registry property is set to a non-null value, the TransientSpace will be configured to bind itself to an RMI registry. This property has no effect on how the service binds itself to Jini lookup services. Note that this property is used only when creating a TransientSpace from the command line.

Note: Support for binding to an RMI registry has been deprecated and will be removed in future versions of TransientSpace. The Jini lookup service is the preferred method for advertising a service.

8.2.12.3 com.sun.jini.rmiRegistryPort

If the RMI registry is being used, this property can assign the port on which the service will look for the registry (1099 is used if this property is not set). This property is only used when creating a TransientSpace from the command line.

8.2.12.4 com.sun.jini.outrigger.spaceName

This property defines the name associated with the TransientSpace. If the space is bound to an RMI registry, then the value of the spaceName property is the name to which it is bound. If the space is bound to a Jini lookup service, this name is used to create an attribute of type net.jini.lookup.entry.Name that is associated with the space's ServiceItem. If this property is not set, a default value of JavaSpace is used. This property is only used when creating a TransientSpace from the command line.

8.2.12.5 com.sun.jini.outrigger.debug

If set, this property enables the output of debugging information. The comma-separated list in the value of the property specifies which subsystems should generate debugging information, and to which output stream that information should be sent. Messages will go to System.out by default, unless an optional log file is provided on a per-subsystem basis. See the manual page at jini1_1/doc/manpages/outrigger.html or jini1_1/doc/release-notes/jsk/serviceproperties.html for a complete description of this property.

8.2.13 System Properties Controlling TransientSpace and Client JVM

The following system properties control the behavior of the proxy objects created by a given TransientSpace server when they are in a client's JVM. Each of these properties are set on the client. If you use these properties, you should ensure that the client grants the appropriate permissions so the TransientSpace service proxy can read these properties.

8.2.13.1 com.sun.jini.outrigger.debug

See the discussion in Section 8.2.12.5, "com.sun.jini.outrigger.debug".

8.2.14 System Properties Controlling FrontEndSpace and Server JVM

8.2.14.1 com.sun.jini.outrigger.backend.vm.options

This property allows extra JVM command-line arguments to be passed to the backend process associated with a FrontEndSpace. The value of this property should be a string that can be passed as an argument to the JVM in which the backend is running.

8.2.14.2 com.sun.jini.outrigger.java

This property specifies what JVM implementation to use for the backend process. The value of this property must be the absolute path to an executable and that executable must accept all the standard JVM commandline options (for example, -D, -classpath, and -Xbootclasspath as documented for the Solaris Operating Environment and Microsoft Windows platform versions of the java command). If this property is not set, a default value of System.getProperty("java.home") + File.separator + "bin" + File.separator + "java" is used, unless the value of the java.home property can't be read or it is null, in which case "java" is used.

8.2.14.3 com.sun.jini.outrigger.backend.gcInterval

This property controls how aggressive the FrontEndSpace will be when garbage collecting the persistent store. The FrontEndSpace writes directives into logs that are consumed by a backend process. This property is set to an integer that represents how many logs must be consumed before the persistent store is garbage collected. Once set to an integer, *N*, the persistent store is garbage collected by the backend process every *N* logs. If not specified, a default value of 5 is used.

8.2.14.4 com.sun.jini.outrigger.basicspace.reapingInterval

This property controls how often the active reaping of unused entry and template objects takes place in the FrontEndSpace. This property is set to a long that represents the time in milliseconds between reaping attempts. If not specified, a default value of five minutes is used. Reducing this value helps reduce the memory footprint of a heavily loaded space.

8.2.14.5 com.sun.jini.outrigger.debug

See the discussion in Section 8.2.12.5, "com.sun.jini.outrigger.debug".

8.2.15 System Properties Controlling FrontEndSpace and Setup JVM

8.2.15.1 com.sun.jini.use.registry

If the com.sun.jini.use.registry property is set to a non-null value, the FrontEndSpace will be configured to bind itself to an RMI registry. This property has no effect on how the service binds itself to Jini lookup services.

Note: Support for binding to an RMI registry has been deprecated and will be removed in future versions of FrontEndSpace. The Jini lookup service is the preferred method for advertising a service.

8.2.15.2 com.sun.jini.rmiRegistryPort

If the RMI registry is being used, this property can assign the port on which the service will look for the registry (1099 is used if this property is not set).

8.2.15.3 com.sun.jini.outrigger.spaceName

This property defines the name associated with the `FrontEndSpace`. If the space is bound to an RMI registry, then the value of the `spaceName` property is the name to which it is bound. If the space is bound to a Jini lookup service, this name is used to create an attribute of type `net.jini.lookup.entry.Name` that is associated with the space's `ServiceItem`. If this property is not set, a default value of "JavaSpace" is used.

8.2.15.4 com.sun.jini.outrigger.debug

See the discussion in Section 8.2.12.5, "com.sun.jini.outrigger.debug".

8.2.16 Java 2 Platform Properties

Every Jini technology-enabled client and service that downloads code will make use of a security manager. For each VM that makes use of a security manager, the `java.security.policy` property must be set to enable a subset of the functionality that is disabled by the security manager.

The 1.1 release of the Jini Technology Starter Kit contains new security policy file templates for each of the services included in the starter kit. These policy files are located in the `jini1_1/policy/` directory.

8.2.17 RMID Properties

The default implementation of `rmid` has changed in the 1.2.2_05 and 1.3 versions of the Java 2 SDK. Now, by default, `rmid` requires that the `java.security.policy` property be set from the commandline. We provide a document describing the impact of this change on activatable services in `jini1_1/doc/release-notes/jsk/execpolicy.html`.

You can also refer to the Solaris Operating Environment manual pages for `rmid` at: http://java.sun.com/products/jdk/1.2/docs/tooldocs/solaris/rmid.html and http://java.sun.com/j2se/1.3/docs/tooldocs/solaris/rmid.html

and the Microsoft Windows platform manual pages for `rmid` at: http://java.sun.com/products/jdk/1.2/docs/tooldocs/win32/rmid.html and http://java.sun.com/j2se/1.3/docs/tooldocs/win32/rmid.html.

8.2.18 RMI Properties

Because most Jini services, including the Jini lookup service and all of the other services in the starter kit are implemented using Java RMI, a number of RMI properties can be set when running a Jini service.

You can refer to the *Properties* section of the RMI specification at http://java.sun.com/products/jdk/1.2/docs/guide/rmi/spec/rmiTOC.doc.html and http://java.sun.com/j2se/1.3/docs/guide/rmi/spec/rmiTOC.html.

8.2.18.1 Supported `java.rmi` Properties

These properties are documented at
http://java.sun.com/products/jdk/1.2/docs/guide/rmi/spec/rmi-properties.doc.html#3601 and
http://java.sun.com/j2se/1.3/docs/guide/rmi/spec/rmi-properties.html#3601

8.2.18.2 Unsupported `sun.rmi` Properties

These properties are documented at
http://java.sun.com/products/jdk/1.2/docs/guide/rmi/spec/rmi-properties.doc.html#3601and
http://java.sun.com/j2se/1.3/docs/guide/rmi/spec/rmi-properties.html#3601

9.0 Administering Jini Software

There is an administration object associated with each service, which can be obtained by calling the `getAdmin()` method. The administration object can be used to examine the contents of a service within the context of a program.

There is a lookup service browser, `com.sun.jini.example.lookup.Browser`, distributed in the starter kit that enables administration of a service though a GUI.

For more information on administration, see the following sections of the Jini Technology Programming FAQ:

10.0 Using Jini Software

10.1 Starting the Jini Software Runtime Environment

Jini technology can be viewed as an infrastructure that provides a runtime environment. This infrastructure is sometimes called a Jini system or a federation where a few infrastructure services must be running before Jini services and clients can be started. That is, before adding a new service or running a client application that would utilize that service, the Jini software runtime environment has to be established.

10.1.1 Infrastructure Services: Lookup Service and RMI Activation Daemon

The infrastructure services include a Jini lookup service and the RMI activation daemon. In order to set up a functioning Jini software runtime environment, you need to start the RMI activation daemon, `rmid`, and a Jini lookup service. Note that the RMI activation daemon must be started before Jini lookup service. The `rmid` command is provided as part of Java 2 platform, while the lookup service implementation is provided as part of the starter kit.

Note: The lookup service is itself a Jini service that runs over RMI. In order to allow clients of the lookup service to download the RMI stub for the lookup service, the lookup service has to be started with a `java.rmi.server.codebase` property. The value of the `codebase` property is a string in the form of a URL, the most common

form of which is HTTP-based. If the codebase property is specified as an HTTP resource, you have to start an HTTP server that can run on a host where the RMI stub of the lookup service resides to serve up the lookup service's stub. The host on which the HTTP server runs is typically the same server on which the lookup service gets started. You can use either an existing HTTP server in your networking environment or the simple HTTP server that is provided as part of the starter kit.

10.1.2 Starting the Infrastructure Services

10.1.2.1 Verify Your Platform and Path Environment

As discussed earlier in this document, you must have installed a supported version of the Java 2 SDK, Standard Edition 1.2.2 or 1.3. To ensure that the combination of operating system and Java 2 SDK, Standard Edition that you are using is one that we have tested, take a look at the list of platforms on which the Jini Technology Starter Kit has been tested, in Section 3.2.2, "Operating Systems and Versions Supported".

You can test to ensure that you have the correct version of the Java 2 SDK installed *and* that your executable path is setup properly by typing:

```
java -version
```

10.1.2.2 Running the RMI Activation Daemon, rmid

For the TCK, the RMI activation daemon needs to be started in the following way:

- On the default port, 1098
- With a log directory to which you have write permission
- In a shell or window with no CLASSPATH setting

If you are using 1.2.2-006 or later of the Java 2 SDK, the security policy for rmid must be configured. For further information, see the document, *rmid exec policies for Jini technology*, at http://developer.java.sun.com/developer/products/jini/execpolicy.html

For example, on a UNIX system with a version of the Java 2 SDK earlier than 1.2.2-006 installed, start rmid as follows:

```
% cd /tmp
% setenv CLASSPATH
% rmid
```

If the Java 2 SDK v1.2.2-006 (or later) is being used, then rmid should be started using:

```
% rmid -J-Dsun.rmi.activation.execPolicy=none
```

which causes rmid to enforce the same security policy as earlier versions of rmid.

To run `rmid` on the Microsoft Windows NT platform, open a Command Prompt window, by using the “Start” button, and selecting “Programs” and then “Command Prompt”. In that window:

```
C:\> cd C:\temp
C:\> set CLASSPATH=
C:\> start /min rmid
```

Again, if Java 2 SDK 1.2.2-006 (or later) is being used, then the last command line above would be replaced with:

```
C:\> start /min rmid -J-Dsun.rmi.activation.execPolicy=none
```

10.1.3 Starting the Infrastructure Services

The starter kit provides commandline utilities for starting these services.They can also be started using the `StartServices` graphical user interface (GUI) program that is provided as part of starter kit. Note that the Jini lookup service must be started *after* the HTTP server and `rmid`.

For detailed instructions on starting these services from the command line or GUI, see:

- GUI – `jini1_1/doc/example/StartingServiceGUI.html`
- Command line – `jini1_1/doc/example/StartingService.html`
- Once these infrastructure services are started, you can then add a new service, such as the Jini transaction manager, or your own custom-built service, to the Jini software runtime environment. Once the services are started, you can run client applications. Detailed information on using the `StartServices` GUI can be found in the online documentation at `jini1_1/doc/example/StartingServiceGUI.html`.

10.2 Running Contributed Jini Service Implementations

In addition to service implementations contributed by Sun Microsystems, such as the Jini lookup service implementation (named Reggie), the starter kit provides other contributed services: a transaction service (Mahalo), a lease renewal service (Norm), a lookup discovery service (Fiddler), and an event mailbox service (Mercury).

Complete instructions and options for starting these service implementations, can be found in the manual page for each service:

- Reggie – `jini1_1/doc/manpages/reggie.html`
- Mahalo – `jini1_1/doc/manpages/mahalo.html`
- Norm – `jini1_1/doc/manpages/norm.html`
- Fiddler – `jini1_1/doc/manpages/fiddler.html`
- Mercury – `jini1_1/doc/manpages/mercury.html`
- Outrigger, transient – `jini1_1/doc/manpages/transientspace.html`
- Outrigger, persistent – `jini1_1/doc/manpages/frontendspace.html`

In addition, there are example docs that describe how to start the services. The page on starting the services from the command line is something of a hold-over from the 1.0 release of the starter kit. As stated earlier in this document, the GUI was completely revised and simplified for the 1.1 release, and is an easy way to run complex command lines and to stop running service.

- GUI – `jini1_1/doc/example/StartingServiceGUI.html`

10.3 Running an Example Service Client

The starter kit contains an example client application, the lookup service browser. Both the binary and source code of the lookup service browser are provided as part of the starter kit.

The browser example is provided to illustrate some of the basic concepts of Jini technology:

- Discovery – The browser, as a client, discovers a lookup service or a group of lookup services.
- Lookup – Once lookup services are found, the browser can query each lookup service for services that have been registered. The browser can also narrow its query criteria by defining a template in which specific service types and attributes are defined.
- Distributed Event – The browser receives notification of newly discovered lookup services and newly added services to the lookup services.
- Leasing – The browser registers its interest of event notification, for example, newly added services, with a lookup service. Each of these registrations has an associated lease. The lookup service sends an event notification to the browser only during the lease period.

  The lookup service browser is an example of a service *client*, and thus does not show how a service gets created and registered into a lookup service.

10.3.1 Lookup Service Browser Operations

By default, when the lookup service browser is started, it searches for all available lookup services. It can also search for either a particular lookup service or lookup services that are members of a certain group.

Once a lookup service is found, it can perform the following operations:

- Display the services currently registered with the lookup service. The display screen automatically updates itself as new services are added or removed.
- Find services that match your search criteria. The search criteria are defined by a search template. The search template could include service type and attributes.
- Display interface types or classes of all services registered with the lookup service.
- Display attribute classes of all services registered with the lookup service.

In addition, you can perform the following administrative operations:

Note: In order to perform these operations, you need to start the lookup service browser with the `-admin` option, described in the next section.

- Add an attribute, modify an attribute, and remove an attribute for a service that is administrable. An administrable service is one that implements the `Administrable` interface.
- Control which lookup groups and lookup locators the service uses for lookup discovery and join. This operation can be performed only to a service that implements the `JoinAdmin` interface.
- Destroy a service. This operation can be performed only on a service that implements the `DestroyAdmin` interface.
- If the service is itself a lookup service that supports the `DiscoveryAdmin` interface, you can control which groups the lookup service is a member of or control which unicast port the lookup service uses for its lookup locator.

For more details, see `jini1_1/doc/example/LookupBrowser.html#USE`.

10.3.2 Starting the Lookup Service Browser

10.3.2.1 Command Line Usage

To start the browser, the command line usage is:

```
prompt> java -cp jar-file  security-policy-property codebase-property
              com.sun.jini.example.browser.Browser
               [-admin]
               [group,...]
```

10.3.2.2 Examples for Starting Browsers

The following examples show how to start the browser:

```
prompt> java -cp /files/jini1_0/lib/jini-examples.jar
  -Djava.security.policy=/files/jini1_0/example/browser/policy
  -Djava.rmi.server.codebase=http://<host>:8080/jini-examples-dl.jar
       com.sun.jini.example.browser.Browser
```

The next example starts the browser in administration mode:

```
prompt> java -cp /files/jini1_0/lib/jini-examples.jar
    -Djava.security.policy=/files/jini1_0/example/browser/policy
    -Djava.rmi.server.codebase=http://host:8080/jini-examples-dl.jar
         com.sun.jini.example.browser.Browser
         -admin
```

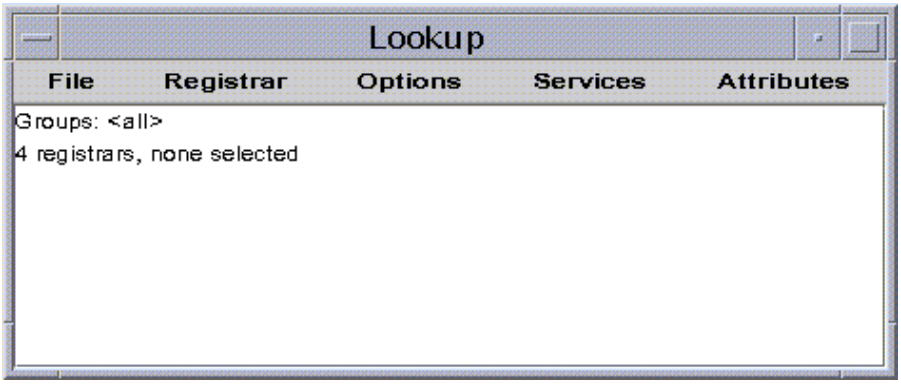
Note: In the examples above, */files* is the directory into which the software was extracted and *host* is the name of the HTTP server that contains the `jini-examples-dl.jar` file. The `-cp` option to the `java` command specifies a class path to search for class files.

10.3.3 Example Screens of the Lookup Service Browser

For detailed instructions on how to use the lookup service browser, see the *Running the Jini Lookup Service Browser* page at `jini1_1/doc/example/LookupBrowser.html`.

10.3.3.1 Starting the Lookup Service Browser without the `-admin` Option

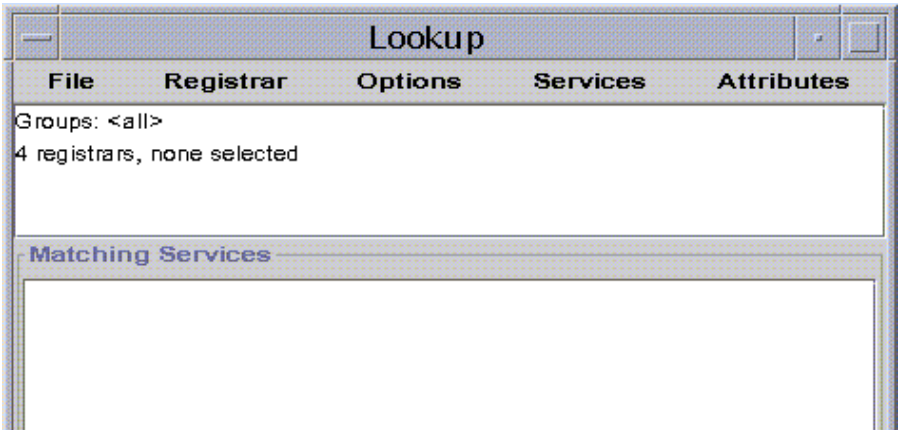
You can start the browser with or without the `-admin` option. Without the `-admin` option, only a single pane appears, as shown below.



10.3.3.2 Starting the Lookup Service Browser with the `-admin` Option

If you start the browser with the `-admin` command-line option, the text area is split into two panes. The upper pane shows groups, templates, and the number of matches and the lower pane shows matching services.

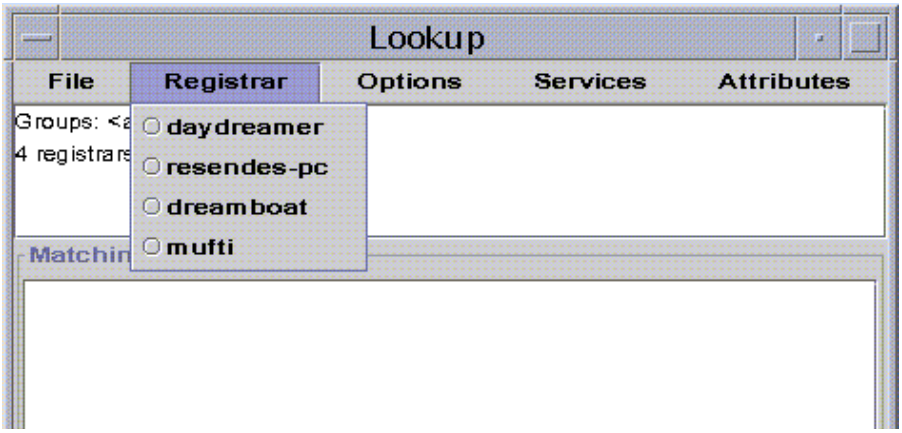
Note: The remaining screens display the browser with the `-admin` option.



10.3.3.3 Displaying the Lookup Services Discovered

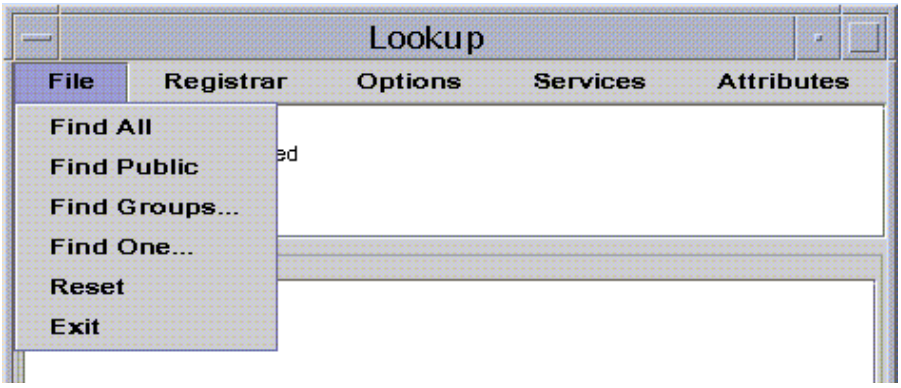
By default, when the browser starts, it searches for all available lookup services. If you specify one or more group names on the command line (separated by commas, with no spaces), the browser only searches for lookup services that are members of those groups. The text area changes to show you how many lookup services have been found.

To select a lookup service for browsing, choose one from the Registrar menu.



10.3.3.4 Discovering Lookup Services

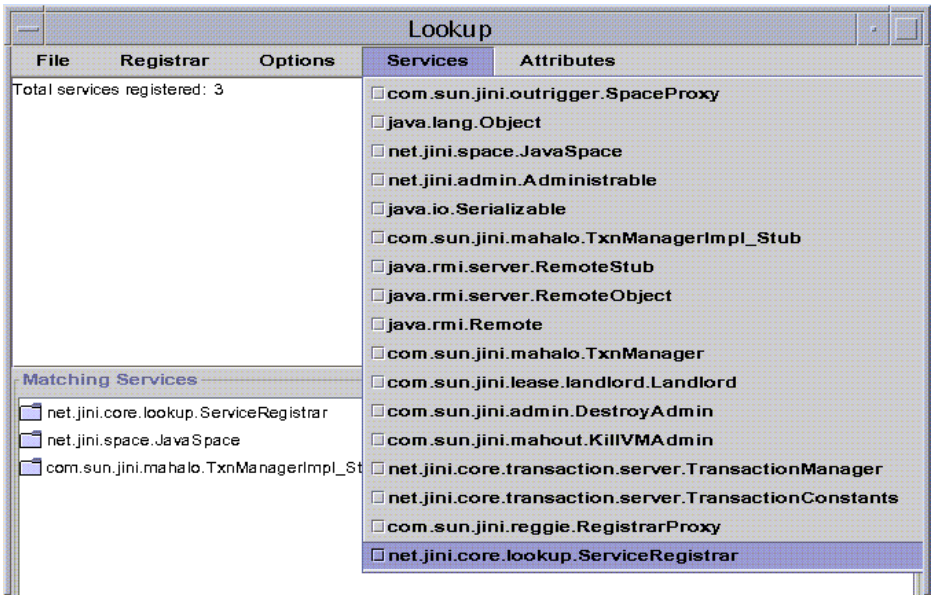
- To restrict the set of lookup services to just the Public ones, that is, those in Group, select the Find Public menu option from the File menu.
- To restrict the set of lookup services to just those in specific Groups, select Find Groups from the File menu, and then enter one or more group names in the pop-up dialog box, separated by spaces or commas. You can use the name public to refer to the Group.
- To restrict yourself to a single Jini lookup service and automatically select it for browsing, select Find One from the File menu, and then enter the name of the host where the lookup service resides, optionally followed by a port number.
- To go back to finding all Jini lookup services, select Find All from the File menu



10.3.3.5 Displaying Services Registered with a Lookup Service

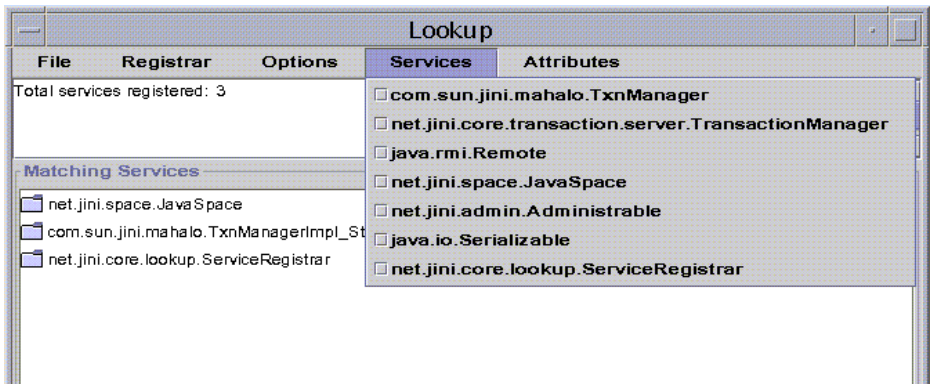
Once you have selected a lookup service for browsing, the text area changes to show the total number of services currently registered with the lookup service. As new services register or existing services unregister, this text changes automatically.

Also note that in the lower pane, administrable services are shown with a blue icon; non-administrable services are shown with a gray icon.



10.3.3.6 Service Types with Default Option

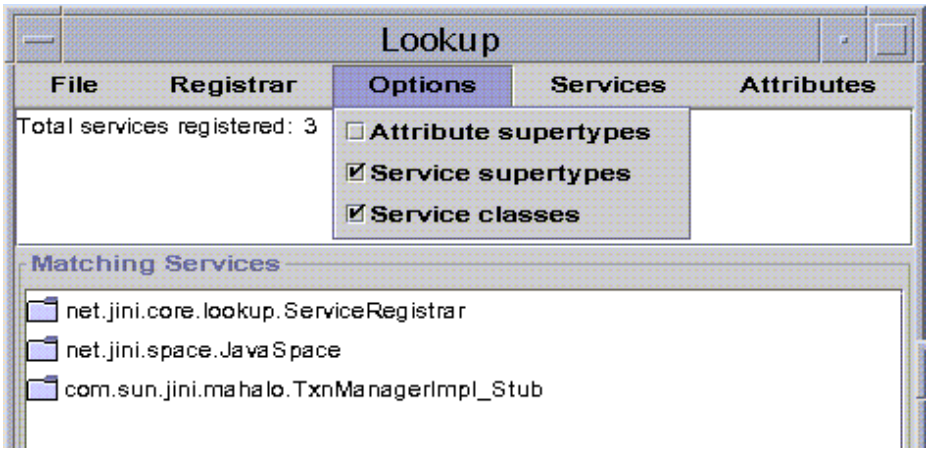
The Services menu shows service types that can be added to your current template to refine your search. When you select one, it appears in the text area to show you what your new template contains, and the revised number of matching services is shown. You can then go back to the Services menu to select additional types, or to de-select existing choices.



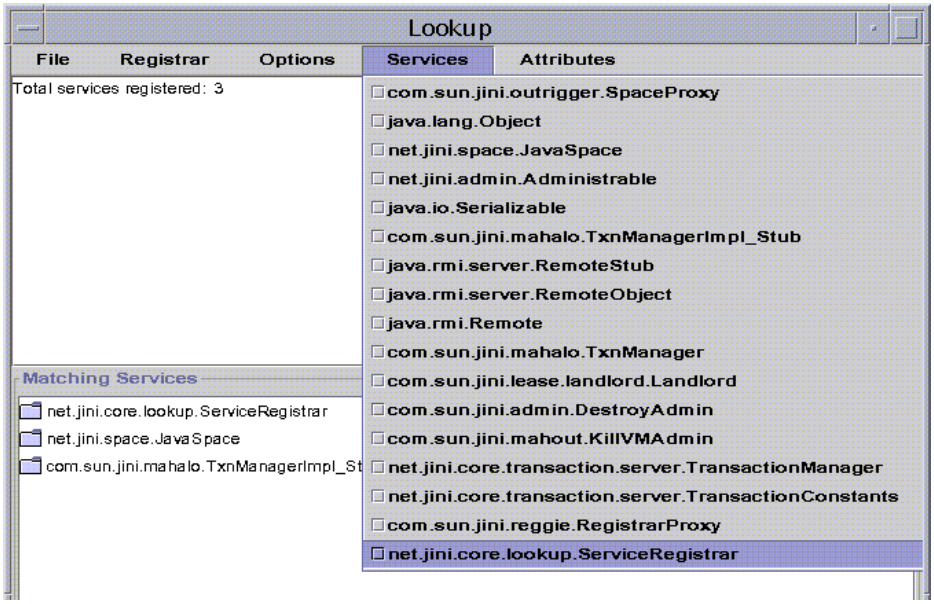
10.3.3.7 Selecting Different Service Types

By default, the Services menu only shows interface types. You can select Service classes from the Options menu to see classes instead. By default, only the most-derived types are shown, but you can select Service supertypes from the Options menu to see

supertypes as well. The following screen shows that you have selected to display all class types including super types.



The following screen displays the result.



10.3.3.8 Displaying Attributes Types

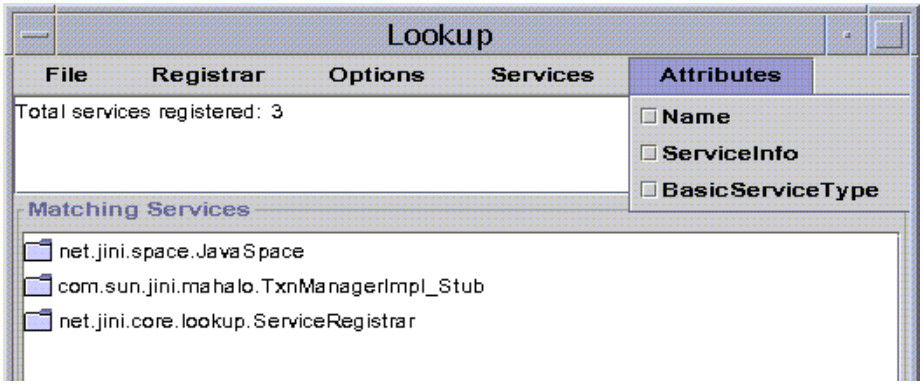
The Attributes menu shows attribute set classes—with package prefixes removed—that can be added to your current template to refine your search. When you select one:

1. The attribute set class appears in the text area to show you what your new template contains
2. The revised number of matching services is shown
3. That menu item changes to a pull-aside menu

You can go back to the Attributes menu to select additional attribute set classes, or traverse a pull-aside menu to see the names of attributes, each of which is itself a pull-aside menu. You can traverse these menus to see the possible values for the attributes, and select one to further refine your search.

You can deselect existing attribute choices, and you can deselect the match menu item to de-select the entire attribute set.

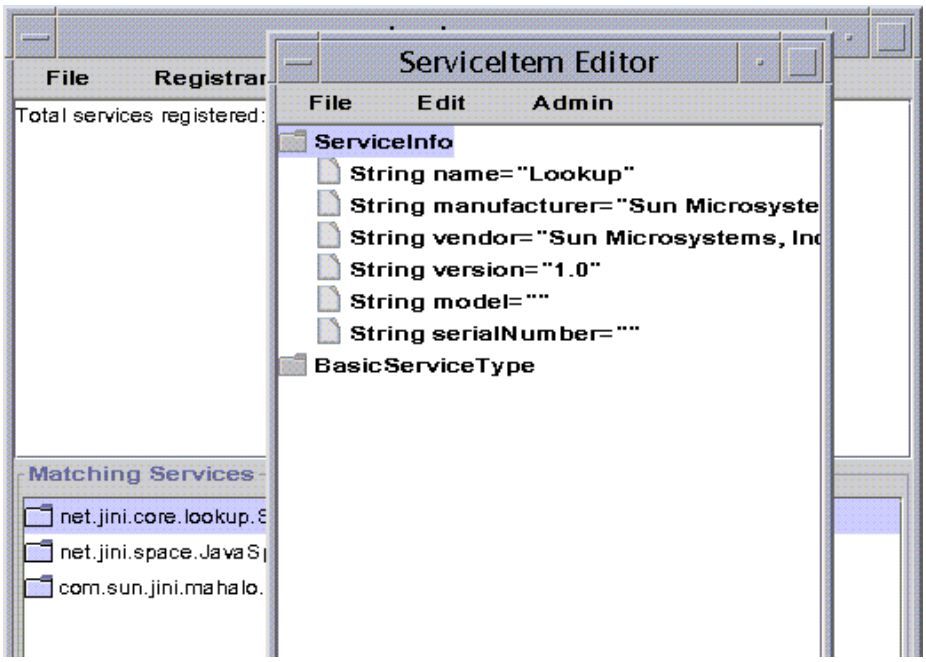
By default, only the most-derived attribute set classes are shown in the Attributes menu, but you can select Attribute supertypes from the Options menu to see superclasses as well. The following screen shows the most-derived attribute set classes.



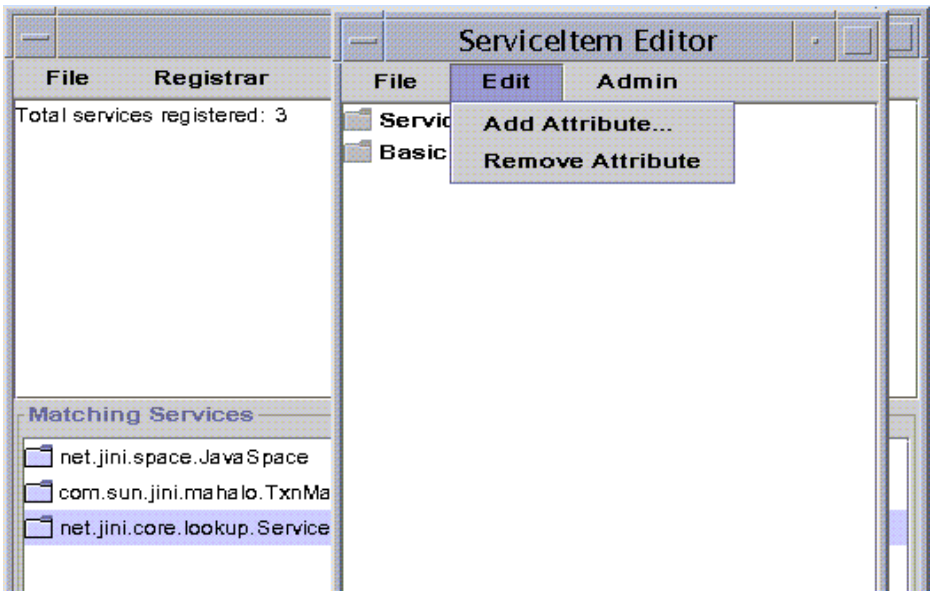
10.3.3.9 Using the ServiceItem Editor

When you start the browser with the `-admin` option, services appear in the lower pane. This pane shows the administrable services with a blue icon and the non-administrable services with a gray icon.

To see details about a particular administrable service, double-click the service. This brings up an editor window, showing the attribute set classes. Double-click an attribute set class to expand the view to include its attribute values, double-click it again to collapse the view. Those attribute sets and attributes with blue icons can be edited, those with gray icons cannot. Double-click on an attribute to change its value.

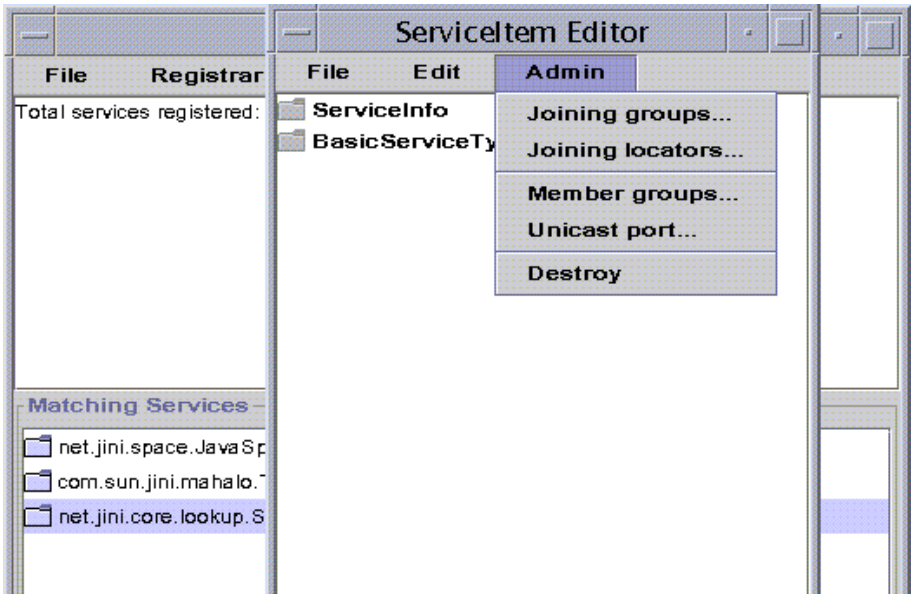


You can also add or remove an attributes from a service as shown in the following figure.

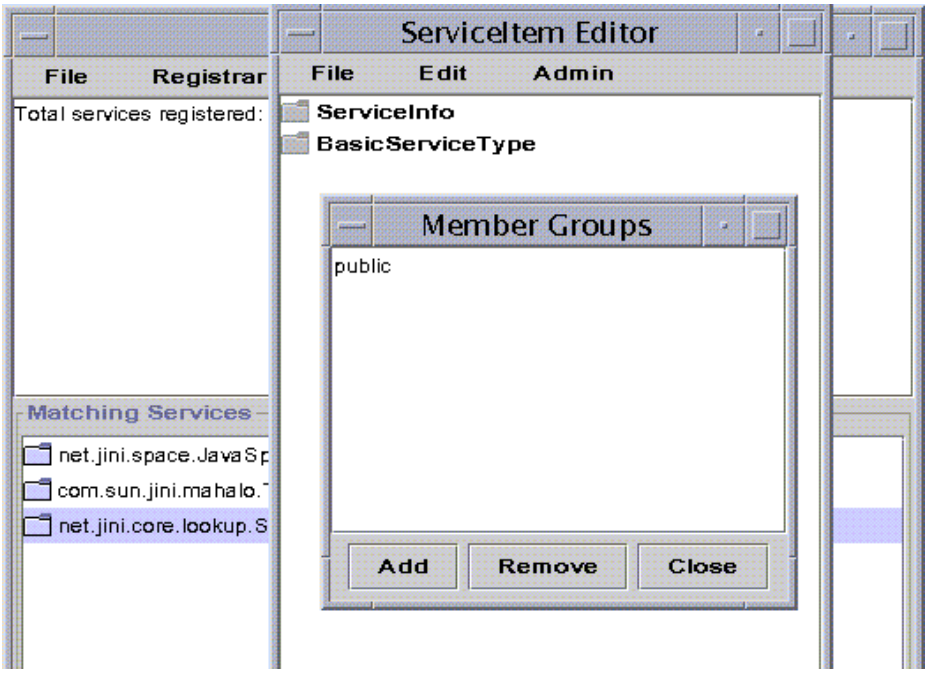


10.3.3.10 Administering Services

- If the service supports the `JoinAdmin` interface, then you can use Joining groups and Joining locators from the Admin menu to control which lookup groups and lookup locators the service uses for lookup discovery and join.
- If the service is itself a lookup service that supports the `DiscoveryAdmin` interface, then you can use Member groups... from the Admin menu to control which groups the lookup service is a member of. You can use Unicast port... from the Admin menu to control which unicast port the lookup service uses for its lookup locator.
- If the service supports the `DestroyAdmin` interface, then you can use Destroy from the Admin menu to permanently destroy the service.



The following screen shot shows that the lookup service is a member of a group whose name is **public**.



10.3.4 Binary and Source Code of Lookup Service Browser

The starter kit provides both binary and source code for the lookup service browser. The binary code is in the `jini1_1/lib/jini-examples.jar` file and the source code is in the directory `jini1_1/source/vob/jive/src/com/sun/jini/example/browser`.

10.3.5 Code Examples of Jini Software Classes and Interfaces in the Lookup Service Browser

The lookup service browser is an example of a Jini service client. The source code gives you a good idea of what and how Jini software classes and interfaces are used by a typical service client application.

- `net.jini.discovery.LookupDiscovery` class – This class is used to discover lookup services which are nearby. Whenever a new lookup service is found, the Browser is notified by this class.
- `net.jini.core.discovery.LookupLocator` class – This class is used to discover a lookup service on a particular host.
- `net.jini.core.lookup.ServiceRegistrar` interface – The `proxy` objects of lookup services are referenced by this interface. The browser invokes methods of this interface in order to perform operations like retrieving a set of services that match a template or registration of “newly added service to the lookup service” event notification.

- `net.jini.core.lease.Lease` interface – The browser uses this interface as a reference to a `lease` object. The browser gets a `lease` object as a return value of event registration to the lookup service.
- `net.jini.lease.LeaseRenewalManager` class – The browser uses this utility class to renew leases of associated services.
- `net.jini.lease.LeaseListener` interface – The browser implements this interface. An instance of that class is passed as an argument when the browser renews its `lease` with a lookup service. The `notify()` method of the class will be invoked by the `LeaseRenewalManager` if the `lease` expires or cannot be renewed.
- `net.jini.lease.LeaseRenewalEvent` class – The browser receives an instance of this class as a parameter to the `notify()` method of the `com.sun.jini.lease.LeaseListener` interface.
- `net.jini.admin.Administrable` interface – The browser checks if a service is an instance of this interface and displays it accordingly with a blue icon in the admin pane.
- `net.jini.admin.JoinAdmin` interface – The browser checks if a service is an instance of this interface and allows further administration by the user if it is, for example, displaying and altering attributes of the service, or displaying which lookup services the service should join.
- `net.jini.lookup.DiscoveryAdmin` interface – The browser checks if a service is an instance of this interface. If it is, the browser assumes the service is a lookup service and controls which groups a lookup service is a member of, and which TCP port a lookup service uses for its lookup locator.
- `net.jini.lookup.entry.ServiceControlled` interface – The browser checks if an attribute of a service is an instance of this interface and displays accordingly.
- `net.jini.core.entry.Entry` interface – The browser uses this interface to refer to instances of Attributes.

10.4 Implementing and Running Your Own Services

Once a Jini software runtime environment is established, you can add your own service into a Jini system so that it can be used by other services or client applications.

10.5 Running the TCK

Complete instructions for running the TCK can be found in the document `jinitck1_1A/doc/running.html`.

11.0 Uninstalling the Jini Technology Starter Kit

11.1 Pre-Uninstall Considerations

Note: Be sure to backup your system before starting any uninstallation.

Stop any running processes, such as the lookup service and `rmid`. Remove the logs created by any of the services (the lookup service, transaction manager, JavaSpaces service, and `rmid`). These log file directories will be subdirectories of the directory in which the commands to invoke the services were executed.

11.1.1 Estimated Time Needed to Uninstall

Note: The time estimate does not include backing up or saving any development files.

No more than a minute or two.

11.2 How to Uninstall the Jini Technology Starter Kit

Because there is no formal installation procedure, there is no formal procedure for removing the starter kit or the TCK -- just remove the appropriate directory.

12.0 Tuning and Troubleshooting

For tuning and troubleshooting beyond what is described here, see Section 9.0 of the *RMI Support Readiness Document*.

12.1 Downloading or Installation Feedback

http://java.sun.com/feedback/faq/downloading.html

12.2 Common User or System Administrator Problems

12.2.1 Missing or Misconfigured Security Policy File

Errors in the policy file usually lead the JVM to ignore the policy file entirely, leading to baffling `SecurityExceptions`. The system property and value `java.security.debug=access,failure,policy` will cause a JVM to produce large amounts of output describing its parsing and application of security policy files, and examination of the result will usually indicate what is wrong.

12.2.2 Codebase Problems

Most codebase problems show up only as mysterious `ClassNotFoundExceptions`. It is very important, then, to understand when a codebase is right. The tutorial at http://java.sun.com/products/jdk/1.3/docs/guide/rmi/codebase.html is useful for understanding how the codebase property is supposed to work.

The HTTP server that is distributed with the Jini Technology Starter Kit should be run with the `-verbose` option, an explicit port number, and an explicit document root to avoid confusion:

```
java -jar /<install_directory>/jini1_1/lib/tools.jar -port 8080
     -dir /<install_directory>/jini1_/lib -verbose
```

The `-verbose` option causes the HTTP server to log attempted downloads of class and JAR files. This type of logging is important because it distinguishes between two types of code downloading failures: asking the correct HTTP server for the wrong file, and not talking to the HTTP server at all.

If a URL is intended to identify a single JAR file, then it should do so in the usual way: `http://hostname:8080/path/jarfile.jar`. The URL must not be abbreviated in any way, except that the port number may be omitted if it is port 80. If the JAR file is used to locate a file that is not part of a Java programming language package, such as a security policy file, then you may have to use the form `http://hostname:8080/path/jarfile.jar!/` to denote that the JAR file should be searched as a directory.

Otherwise, if a URL is intended to identify a file system directory from which individual class files will be downloaded, the URL must end in a slash character ('/'). The indicated directory must be the root of the package directory of the compiled (`.class`) files. For example, if the URL `http://hostname:8080/com/foo/bar/Thing.class` will download the class `Thing` in the package `com.foo.bar`, then the codebase URL must be "`http://hostname:8080/`", rather than "`http://hostname:8080`", "`http://hostname:8080/com/foo/bar/`", or any other variants.

12.3 Common User Questions

- Marketing and Jini technology product idea questions – http://sun.com/jini/faqs/index.html
- Licensing questions – http://www.sun.com/jini/faqs/index.html;$sessionid$NTUAFMIAAABXRAMTA1LU5YQ#licensing

12.4 Product Limitations

12.4.1 Security

Currently, Jini technology relies on the security of the Java 2 platform. While work is underway to add authentication and authorization to RMI that builds upon JAAS (see http://java.sun.com/products/jdk/rmi/rmisec-doc/intro.html), that work won't be part of the Java 2 SDK until the release of Java 2, 1.4. When RMI security is released, Jini technology is expected to use the security mechanisms provided in it.

12.4.2 Outrigger Runs in a Single JVM

The Outrigger implementations offer a JavaSpaces service which is centralized rather than distributed. That is, there is a single Java virtual machine (JVM) that hosts the space; if the JVM crashes, the space is unavailable until the JVM restarts. While there are no known limitations precluding the creation of a JavaSpaces service that spans more than one JVM, Sun has not yet provided such an implementation in the starter kit.

12.5 Error Message Guide

- Descriptions of Jini technology exceptions – `jini1_1/doc/api/index.html`
- RMI exceptions – `http://java.sun.com/products/jdk/1.3/docs/api/index.html`

12.6 Known Issues and Their Workarounds

Some of the known issues are described in the release notes of the JXP and the JSK.

- Known issues for the JXP – `jini1_1/doc/release-notes/jxp.html`
- Known issues for the JSK – `jini1_1/doc/release-notes/jsk.html`

At the time of the Jini Technology Starter Kit 1.1 release, there were no known P1, P2, or P3 bugs.

12.7 Bug Categories in Bugtraq

- net_jini_core
- net_jini_admin
- net_jini_core
- net_jini_discovery
- net_jini_entry
- net_jini_event
- net_jini_lease
- net_jini_lookup
- net_jini_space
- com_sun_jini_admin
- com_sun_jini_benchmark
- com_sun_jini_benchmark_lookup
- com_sun_jini_collection
- com_sun_jini_constants
- com_sun_jini_debug
- com_sun_jini_discovery
- com_sun_jini_example
- com_sun_jini_fiddler
- com_sun_jini_lease
- com_sun_jini_lookup
- com_sun_jini_mahalo
- com_sun_jini_mahout
- com_sun_jini_mercury
- com_sun_jini_norm

- com_sun_jini_proxy
- com_sun_jini_reggie
- com_sun_jini_reliableLog
- com_sun_jini_system
- com_sun_jini_thread
- com_sun_jini_test_clm
- com_sun_jini_test_discovery
- com_sun_jini_test_lookup
- com_sun_jini_test_misc
- com_sun_jini_test_util
- com_sun_jini_test_txnmanager
- build
- doc
- other

12.8 debug Properties

Each of the services in the starter kit has a debug property that can be used to enable the output of debugging information. To activate it, the value of the property must be set to a comma-separated list of keywords. Each keyword activates a different set of execution traces, and the set of keywords for one service are different from the set of keywords for a different service. The keywords are described in each service's manual page.

- Reggie – `jini1_1/doc/manpages/reggie.html`
- Mahalo – `jini1_1/doc/manpages/mahalo.html`
- Norm – `jini1_1/doc/manpages/norm.html`
- Fiddler – `jini1_1/doc/manpages/fiddler.html`
- Mercury – `jini1_1/doc/manpages/mercury.html`
- Outrigger, transient `-jini1_1/doc/manpages/transientspace.html`
- Outrigger, persistent `-jini1_1/doc/manpages/frontendspace.html`

13.0 Reference Information

13.1 Product Information

- Jini Technology Home Page – http://www.sun.com/jini/

13.2 Technical Documentation

13.2.1 Technical Articles on Distributed Computing

- *A Note on Distributed Computing* by Jim Waldo, Geoff Wyant, Ann Wollrath, and Sam Kendall – http://www.sun.com/research/technical-reports/1994/abstract-29.html

13.2.2 Specifications for the Jini Technology Core Platform

http://sun.com/jini/specs/ or jini1_1/doc/specs/jcp.html

The following documents are included in the starter kit in PostScript and PDF format, and online in HTML, PostScript and PDF:

- *Jini Architecture Specification*
- *Jini Technology Core Platform Specification*
- *Jini Device Architecture Specification*
- *Jini Technology Glossary*

13.2.3 Specifications for the Jini Extended Platform

http://sun.com/jini/specs/ or jini1_1/doc/specs/jxp.html

The following documents are included in the starter kit in PostScript and PDF format, and online in HTML, PostScript and PDF:

- *A Collection of Jini Technology Helper Utilities*
- *JavaSpaces Specification*

13.2.4 Jini Technology Books

http://www.sun.com/jini/books/

13.2.5 Jini Technology Fact Sheet

http://www.sun.com/jini/factsheet/

13.2.6 Jini Technology API Documentation

jini1_1/doc/api/index.html

13.2.7 RMI Resources

http://java.sun.com/products/jdk/rmi/

13.3 Additional References

13.3.1 Official Jini Technology Web site

http://www.sun.com/jini/

This page contains links to the following sites:

- Jini Technology Executive Overview – http://sun.com/jini/overview/
- Jini Community Licensing – http://sun.com/jini/licensing/

- Jini Community Program Overview – http://sun.com/jini/community/
- Jini Technology News and Articles– http://sun.com/jini/news/index.html

13.3.2 Sun's Logo and Trademark Guidelines

http://sun.com/policies/trademarks/

13.3.3 jini-users Newsgroup Posting Archive

http://archives.java.sun.com/archives/jini-users.html

13.3.4 jini-users Mailing List Subscription

http://sun.com/jini/subscribe.html

13.3.5 Jini Technology Programming FAQ

http://www.artima.com/jini/faq.html

13.4 Contacts

- Jini trademark and logo questions – Contact Sun Microsystems Marketing at jini-info@sun.com and Sun's Trademark Department at trademarks@sun.com
- Licensing model questions/comments – Contact SCSL-comments@java.sun.com or jini-commercial@sun.com
- Technical questions – Contact jini-comments@java.sun.com or jini-users@java.sun.com
- General questions about Jini technology – Contact jini@sun.com